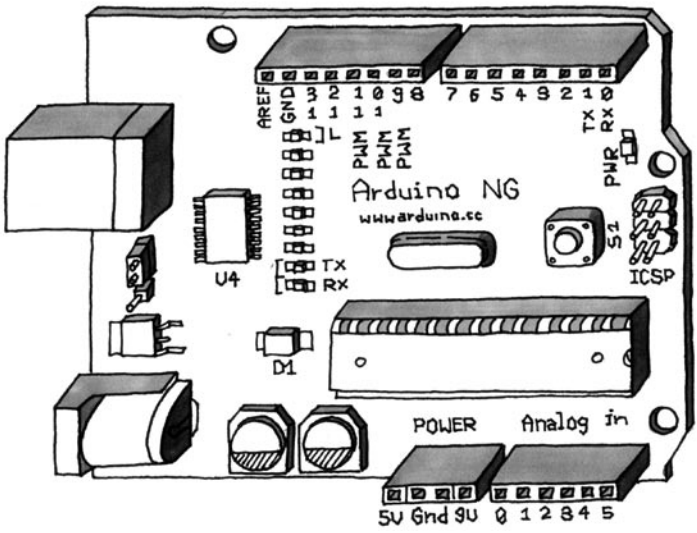
Getting started with

ARDUINO

BETA Version



Written by Massimo Banzi.
With materials written by Massimo Banzi, Erica Calogero, David Cuartielles, Jeff Gray, Tom Igoe, David Mellis and Cristian Nold.
Illustrations by Elisa Canducci.

Thanks

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:

Aliadi Cortelletti for typesetting the booklet.

Barbara Ghella, Stefano Mirti, Claudio Moderini.

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

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 preassembled; the open-source IDE can be downloaded for free.

Arduino is different from other platforms that can be found on the market because of these features:

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 EUR 20 and replacing a burnt out chip on the board is easy and costs no more than EUR 5. So you can afford to make mistakes.
There is an active community of users so there is plenty of people who can help you.

What does this all mean? We're going to discover it this through this booklet that is designed to help designers and artistis to understand what benefits they can get from learning how to use the Arduino platform and adopting its philosophy.

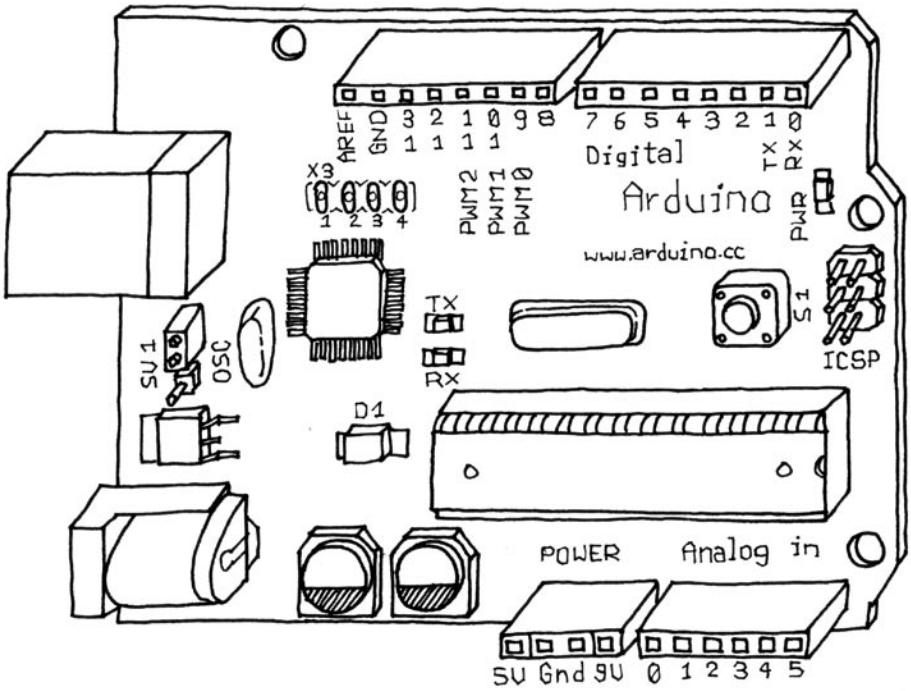
/ what is interaction design?

There are many definitions of Interaction Design but the one that I like the most is simply "Interaction Design is about the design of any interactive experience". In today's world this generally is about the creation of meaningful between us (humans) and artefacts. We also like to explore the creation of beautiful and maybe even controversial between technology and us.
Specifically we believe in designing through an iterative process based on prototypes of ever increasing fidelity. This approach ,also part of some types of "conventional" design, can be extended to include prototyping with technology and in particular with electronics.
This particular brand of Interaction Design is called Physical Computing (or Physical Interaction Design). This booklet is in no way a substitute for a book on Physical Computing, we recommend you buy Tom Igoe's excellent "Physical Computing" book.

/ what is physical computing?

Physical Computing is about prototyping with electronics, turning sensors, actuators and microcontrollers into materials for designers and artists.
It involves the design of interactive objects that can communicate with humans using sensors and actuators controlled by a behaviour implemented as software running inside a microcontroller.

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 (small computers on a single chip) have become cheap and easier to use allowing the creations of better tools.
The work that we have done with Arduino is to try to bring these tools one step closer to the beginner allowing people to start building stuff after only 3 or 4 days of workshop.
With Arduino the designer or artist can get to know the basics of electronics and sensors very quickly and start building prototypes with an investment of as little as 70EUR.



This is a picture of the Arduino board. At the beginning you might be a bit confused with all those connectors. Here is an explanation of what every element of the board does:
14 Digital IO (pins 0 - 13,) can be inputs or outputs as set in software.

6 Analogue 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 Analogue Out (pins 9, 10, 11) these are actually 3 of the digital pins that can be reassigned to do analogue output.

The board can be powered from your USB port or from the power socket. This setting can be changed with the jumper marked SV1 in the diagram. If the jumper is closest to the USB plug then the board is powered from there. If the jumper is on the 2 pins closest to the DC connector then it is powered from there.

/ the software (IDE)

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 quite important because it's where Arduino is making your life simple and hiding away as much as possible of the complexities of programming microcontrollers.

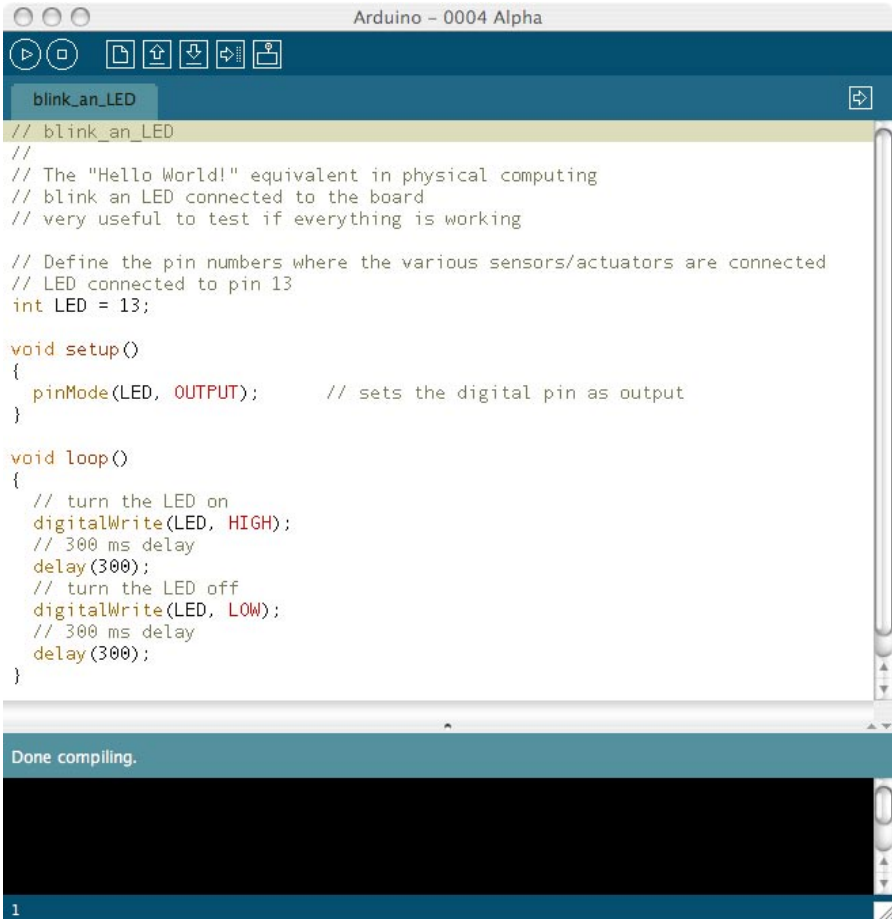
Downloading and installing the software
In order to program the Arduino board you need to download the development environment (IDE) from here:
http://www.arduino.cc/en/Main/Software
Choose the right version for your operating system.
Download the file and uncompress it.
The first thing to do is to install the drivers that allow your computer to talk to your board through the USB port.

Macintosh:
Look for the "Drivers" folder inside the "arduino-0004" folder and 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 apre 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 phase is over you can plug the board into the computer.

Windows:
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.

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



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

Macintosh:
From the "Tools" menu select "Serial Port" and select the port that begins with "/dev/cu.usbserial-". The last 3 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]

Windows:
On Windows the process is a bit complicated at the beginning.
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]
Note: 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.

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.

REALLY GETTING STARTED WITH ARDUINO

Now that we have introduced 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. At the end of these exercises you are 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’s 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. In our body the eye, for example, converts light into signals that get sent to the brain using nerves while in the electronic we could use a simple device called LDR that can measure the amount of light that hits it and report it back as a signal that can be understood by the processor. [PIC LDR.tiff]
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. For example in 0ur bodies, 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. [PIC motor_fan.tiff]
In the next chapters we will see how to read sensors of different types and control different kinds of actuators.

/ basic introduction to programming

Some of you may never have programmed before, if you have you may skip this part. 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 a bit more close to human languages than instructions like

```
lda 0x0f
```

Let's see how we would go about translating a simple behaviour into a piece of program.
A simple programme can always be mocked up as sentence, for example:
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 init() {
// put code here
}

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

At the beginning of the program we declare variables, areas of memory used to store data. Then the void init() function is used to setup the program (define which pins on the processor are inputs and which one are outputs etc)
The last function, void loop(), will then be executed 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.
The text beginning with // is a comment, it's very useful for you to remind yourself what your code does when you re-open it after a while or for other people

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 need, to go into more detail, see: [URL of arduino reference]

Flow control

If [condition] Then

In Arduino, an if statement looks like this:

```
if(expression)
{
  statement;
  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

(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.

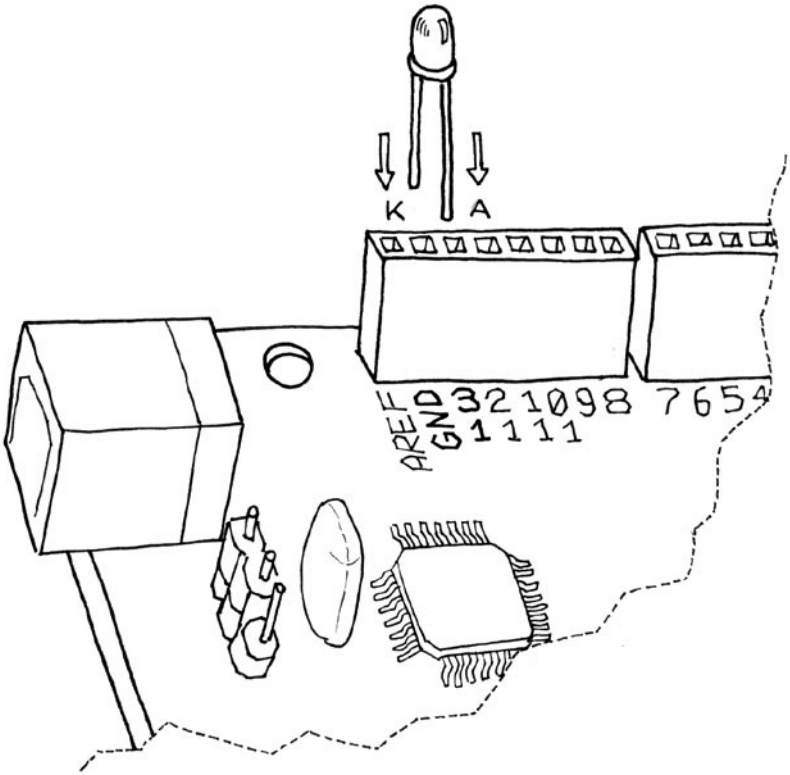
Note In Arduino you don't need to initialise the local variable i. However in java-based languages like Processing you add int in front of the first instance of i.

Delays

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

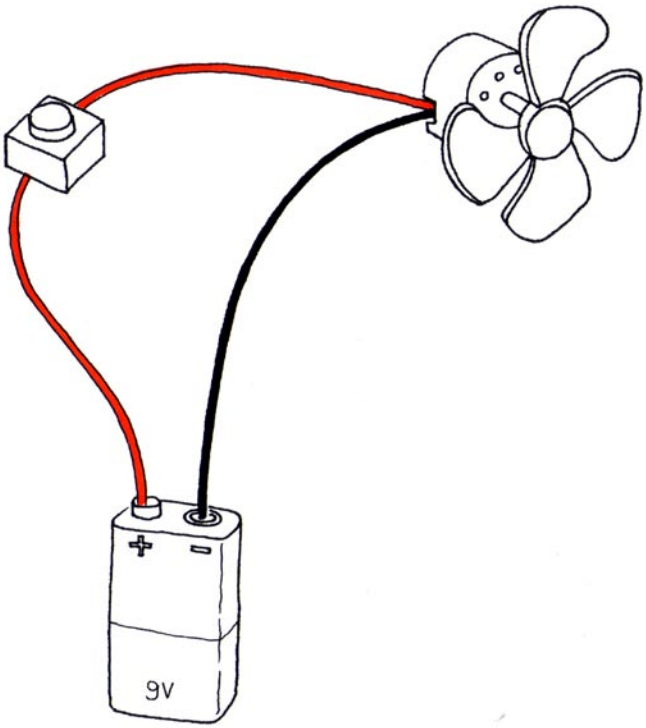
The delay() function is useful for embedded programming to slow down the rate at which the processor updates. This is used to make thing happens at a certain rate, for example if we want to blink and led every second, after we turn it on we can place a delay(1000) which will make the processor sit there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

Assuming that program has been uploaded correctly connect an LED to the pins 13 and GND on the board like you see in the illustration.



/ what is electricity

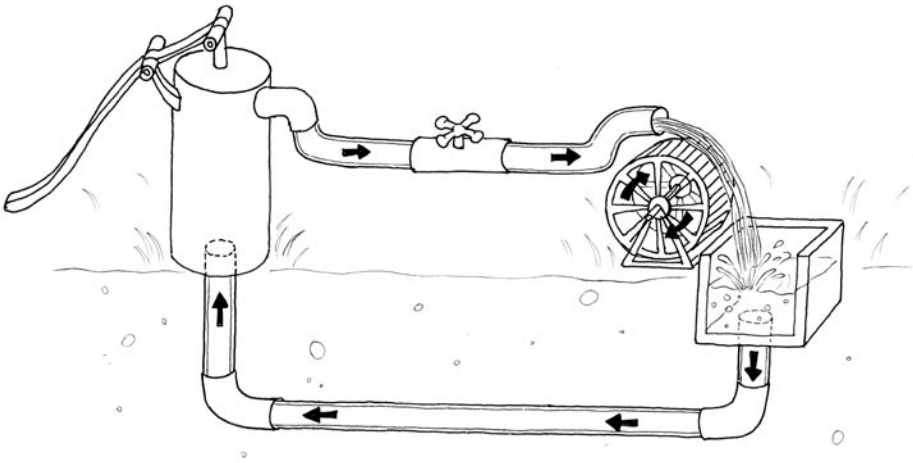
If you have ever done any plumbing at home, electronics won't be a problem for you to understand. Jokes aside, in order to understand how electricity and electric circuits work the best way is to build a mental model called the "water analogy". Let's take a simple device like a portable fan,



if you take it apart you will see that it contains a small battery a couple of wires going to an electric motor and one of the wires is interrupted by a switch. Now makes sure you have a new battery fitted in the device and activate the switch; the motor will start to spin providing the necessary refreshment. How does this work? Well imagine that the battery is a water pump and the switch is a tap while the motor is one of those wheels you see in watermills, when you open the tap water will flow from the pump and push the wheel into motion.

Now in this simple hydraulic system two parameters are important: the pressure of the water (this is given from how powerful is the pump) and the amount of water that will flow in the pipes (this depends from the size of the pipes and the resistance that the wheel will oppose to the stream of water hitting it).

You quickly understand that if you want the wheel to spin faster you need to increase the size of the pipes (but this works only up to a point) and increase the pressure that the pump can achieve. Increasing the size of the pipes allows more flow of water to go through them, effectively by making them bigger we have reduced the resistance they oppose to the flow of water. This works until a certain point where the wheel won't spin any faster because the pressure of the water is not strong enough and this is when we need the pump to be stronger.



This can go on until the point when the wheel falls apart because the water flow is too strong and destroys it. Another thing you will notice is that as the wheel spins the axle will heat up a little bit, this is because no matter how good is the way we have mounted the wheel the attrition between the axle and the holes it is mounted in will generate heat. This is important to understand that in a system like this not all the energy you pump into the system will be converted into movement, some will be lost in a number of inefficiencies and will generally show up as heat emanating from some parts of the system.

So what are the important parts of the system as we described it before? The pressure produced by the pump is one, the resistance that the pipes and wheel oppose to the flow of water and the actual flow of water (let's say that this is represented by the number of litres of water that flow in one second) are the others.

Without going into much details electricity works a bit like water, you have a kind of pump (any source of electricity like a battery or a wall plug) pushes electric charges (imagine them like "drops" of electricity) down pipes represented by the wires 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.

So when you read on a battery 9V you can 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 got and 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, yes you guessed right, resistance and it's measured in Ohms from the German physicist Ohm.

Mr ohm is also responsible for coming up with the most important law in electricity and the only formula you will really need to remember.

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

It's very intuitive if you think about it: 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. Going back to the water flowing in pipes, given a certain pump if I place a tap (which we can assimilate to a variable resistor in electricity) the more I close the tap, increasing resistance to water flow, less water will flow through the pipes. Mr Ohm summarised his law into this formulas:

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

V = R * I

I = V / R

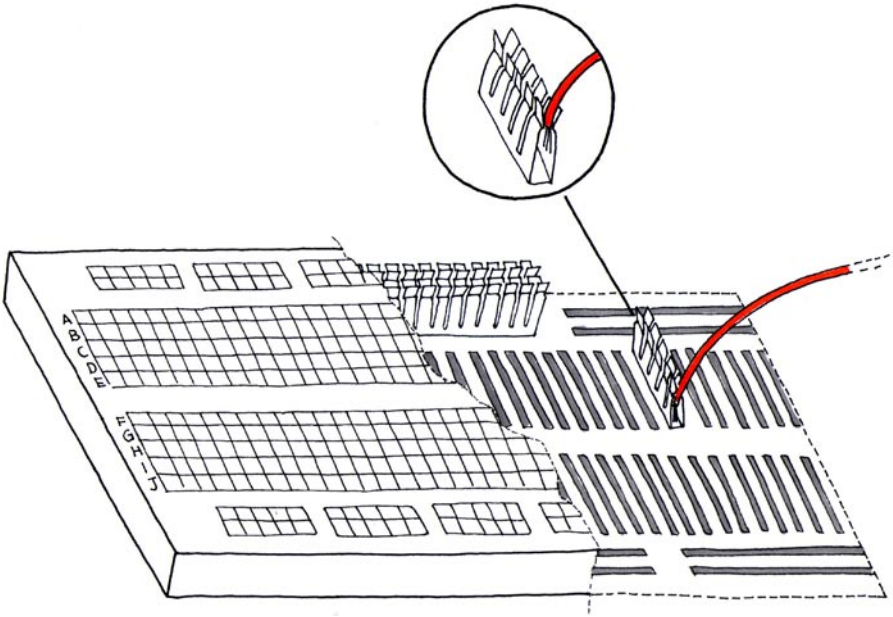
This is the only rule that you really have to memorise and learn to use, because in most of your work this will be the only one you will really need.

/ 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 want 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 clearly rules out soldering, it's a time consuming procedure that puts every component under stress every time you heat them up and cool them down.

The answer to our problems comes from a very practical device called "Solder-less Breadboard".



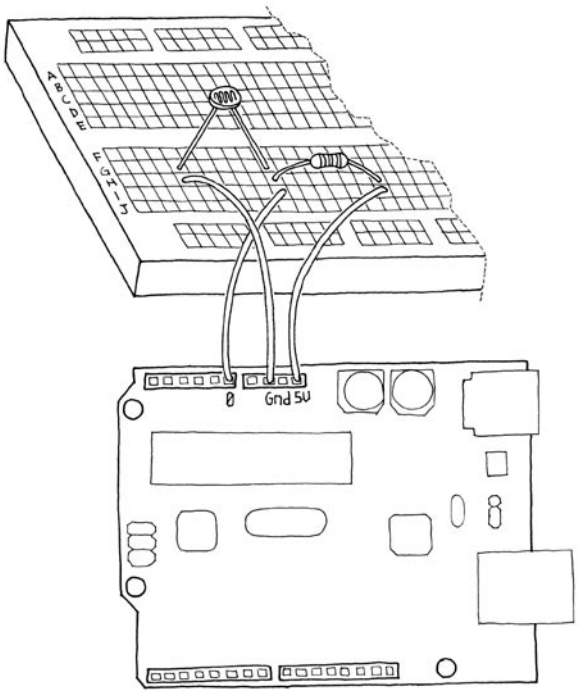
As you can see from the picture it's a small plastic board full of holes, each one of them contains a spring-loaded contact. You can push a component's leg into one of the hole and it will establish an electrical connection with all the other holes in the same vertical column of holes. Each hole is at a distance of 2.54 mm distance from the others, since most of the components have their legs, known to techies as pins, are spaced at that standard distance therefore chips with multiple legs will fit nicely. Not all the contacts on a breadboard are created equal, there are some differences: 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 circuits) The last this you need to know about breadboards is the in the middle there is a large gap that is as wide as the size of a small chip. This shows that the each vertical line of holes is interrupted in the middle so that when you plug a chip you won't short circuit pins that are on the two sides of the chip, clever eh?

/ analogue inputs

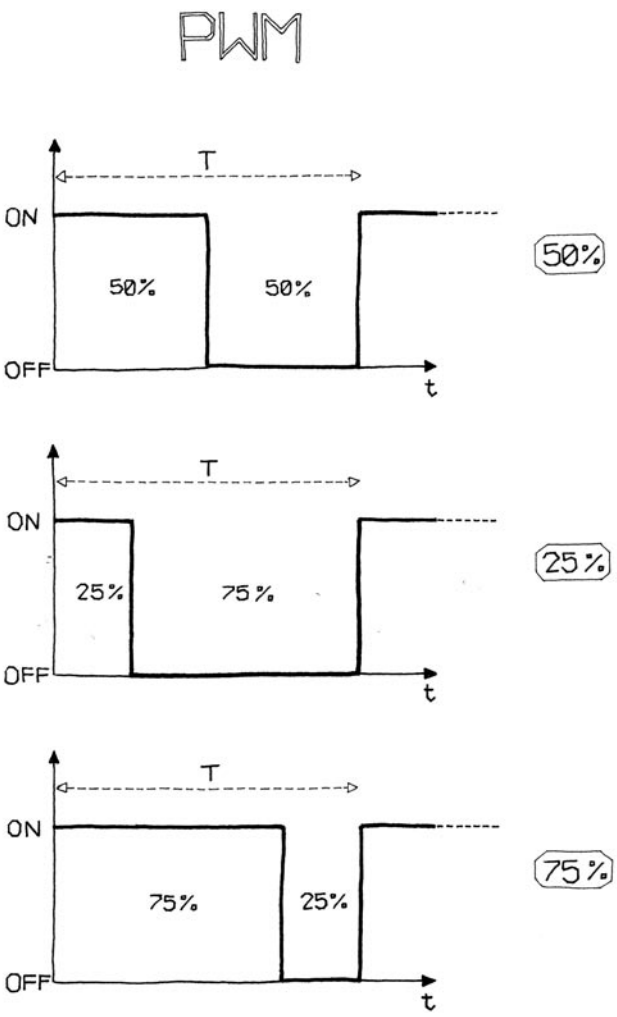
As we have seen in the previous section Arduino is able to detect if there is a voltage applied to one of its pins and report it trough the digitalRead function. This is fine in a lot of applications but the light sensor that we have used before it's also able to tell us not just if there is light or not, it's also able to tell us 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 0 writing analogRead(0) will return 512 etc etc.

If you now build the circuit that you see in the illustration by using a 10k or 4.7k resistor and you run the piece of code you find here you'll see the led blinking at a rate that's dependent on the amount of light that hits the sensor.



/ analogue output (PWM)

With the knowledge that you have accumulated until now you could take up the task of building an interactive lamp, a lamp that can be controlled not just with a boring on-off switch but maybe in a bit of more poetic way.
One of the limitations of the blinking LED examples we have seen until now 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 little trick that makes a lot of things like TV or Cinema possible: Persistance of Vision.



Take the first LED blink example we have seen and now change the numbers in the delay function until you don't see the LED blinking any more. You will see 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 ¼ 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, a fancy name to say that if you blink the LED fast enough you don't see it blink any more but you can change its brightness by changing the ration between the on time and the off time. Here is a small diagram showing how this works.
This technique also works with other devices than the LED. For example you can change the speed of motor in the same exact way.
While experimenting you will see that blinking the LED by hand is a bit inconvenient because as soon as you want to read a sensor or send data on the serial port the LED will flicker. Luckily enough 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.
For example 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 might their light and make light of any colour you like!