Hardware hacking made easy

Any old geek can program software, but it takes some serious commitment to computers to want to program hardware. We show you how to get started…

Arduino is cool. It's cool because it's a tiny device – about three inches by two inches – that comes with a USB port and a programmable chip. It's cool because you can program it using a very simple programming language known as Wiring. But most of all, it's cool because the entire reference design for the hardware is available under a Creative Commons licence, so you're free to build your own if you want to. However, that's probably a little extreme for most people, which is why you can also buy pre-build Arduino boards that are ready for action, and they are available at very low prices too. The Arduino programming IDE is available under the GPL for multiple platforms, so the only things standing between you and your own pet hardware project are an Arduino board, a cool idea, and of course a Box O' Tricks – some neat little parts you can plug into the Arduino to make it do more interesting things.

Electronics and safety

Please remember that electronic parts can be very sensitive: static electricity can be fatal to your Arduino and many other small computing parts, so don't shuffle your feet and wear anti-static wrist bands. Equally, though, remember that electronic parts can be dangerous to you – they often have lead in them, so make sure you wash your hands thoroughly after working with your kit!

Getting set up

Arduino boards come in several varieties, but the main three are the Arduino NG ("next generation"), the Arduino NG Plus, and the Arduino Diecimila. These aren't competing boards – rather the NG came first, and was replaced by the NG Plus, then the Diecimila. We have an NG because we bought it a while ago, but if you try to buy one now you'll probably only find the Diecimila for sale. Ultimately the difference between these boards is very small: the NG Plus has a ATmega168 CPU compared to the ATmega8 found in the plain NG, and the Diecimila has been tweaked to make it fractionally easier to load programs. The ATmega8 and ATmega168 chips are very similar, with the primary difference being that the 168 has space for larger programs, but that won't be a problem for our purposes here – any of those three boards are suitable for following this tutorial.

As for where to get your Arduino board, we recommend PCB Europe (**http://pcb-europe.net**) – they have the Diecimila on sale there for 22 Euros (£15/$31 at the time of writing), and they are very happy to answer questions if you're unsure precisely what you want.

Now comes the slightly more complicated part: what do you want to plug into your Arduino? By itself, the board has 14 digital

› A USB type 2 (left) to USB type 1 (right) cable is required to connect the Arduino to your PC.

connectors and a further 6 analogue connectors, as well as a built-in LED and a reset button, so you need to buy extra parts to make it do things. If you're in the US, you're probably lucky enough to have a Radio Shack in close driving distance, and you'll find a mine of cool stuff there ready to play with – just talk to one of the store assistants, explain you're trying your first electronics projects, and you should be able to walk away with a basketful of goodies for $30 or so. If you're in the UK, your best bet is to place an order at **www.maplin.co.uk** for the parts you want – although I recommend you throw in some of their wonderful Lucky Bags,

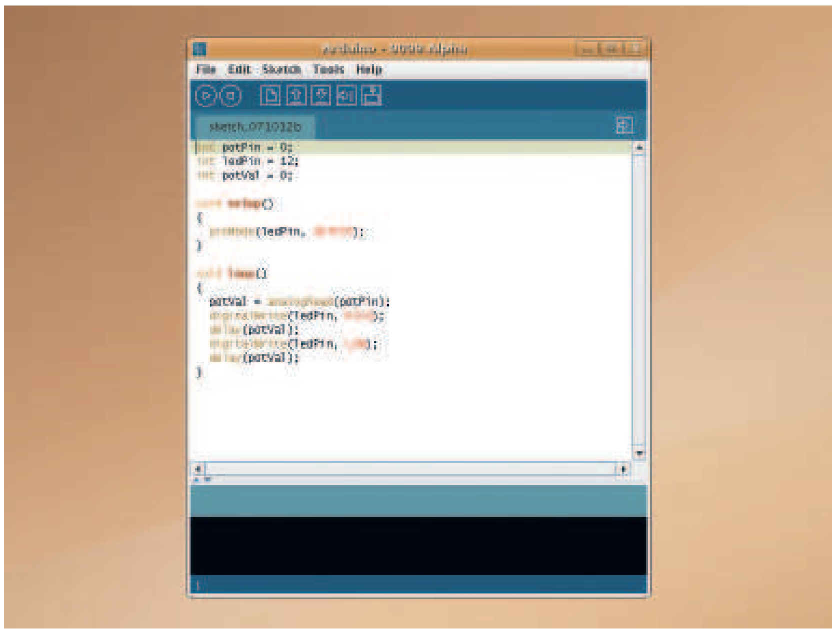
> **"The first thing with any programming project is usually akin to performing a 'hello word' test."**

because it's surprising the ideas that you can get when you are given random parts to work with! Outside of the US and the UK, you can either try to track down a good electronics store in your country (and by "electronics" we don't mean "sells iPods"!), or you can go back to PCB Europe and buy one of its Arduino parts kits.

No matter which option you take, you need at least the following if you want to complete this Arduino primer:

» **A solderless breadboard**. This is usually a rectangular white block of plastic, with lots of holes in, and carries current so you can hook things together.
» **Some jumper wires**. You can buy these pre-cut at different lengths, or make them yourself – it's all the same.
» **A Type A to Type B USB connector**. In normal speak that's standard USB on one side, and the fat USB style on the other side.
» **Some LEDs**. Be colourful!
» **Resistors of various strengths**. You won't need anything more than 10K Ohm resistors, but these are so cheap that you might as well buy a selection.
» **A potentiometer.**
» **A light-dependent resistor.**

You should be able to buy all the above for £10 and still have change afterwards, so don't be afraid to buy a few extra parts to have some fun later!



```
int potPin = 0;
int ledPin = 12;
int potVal = 0;

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

void loop()
{
  potVal = analogRead(potPin);
  digitalWrite(ledPin, HIGH);
  delay(potVal);
  digitalWrite(ledPin, LOW);
  delay(potVal);
}
```

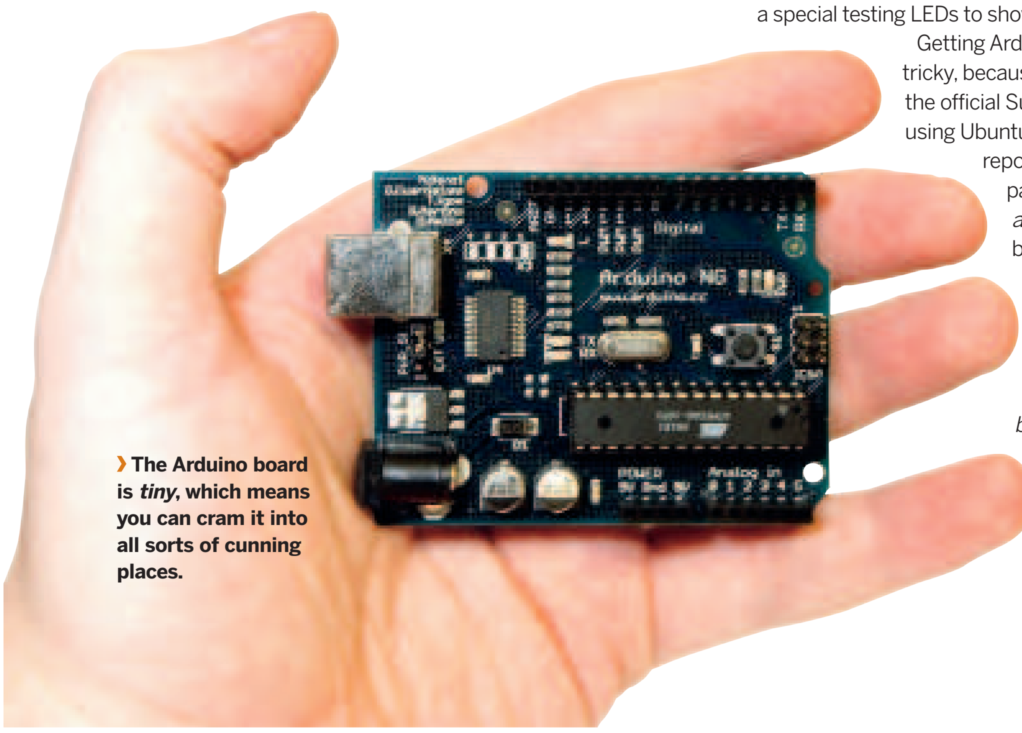
› The Arduino IDE is very simple, offering code highlighting, save and load, and not much more – but you can upload to your board straight from here.

Blinkenlights

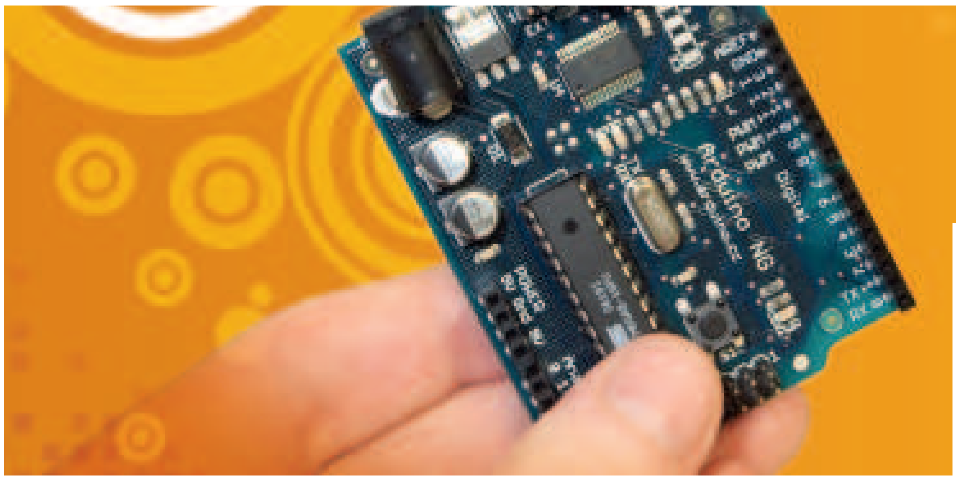
The first thing to do with any programming project – hardware or otherwise – is usually something akin to "hello world": make your program send a simple message to the outside world to show that you have everything configured correctly.

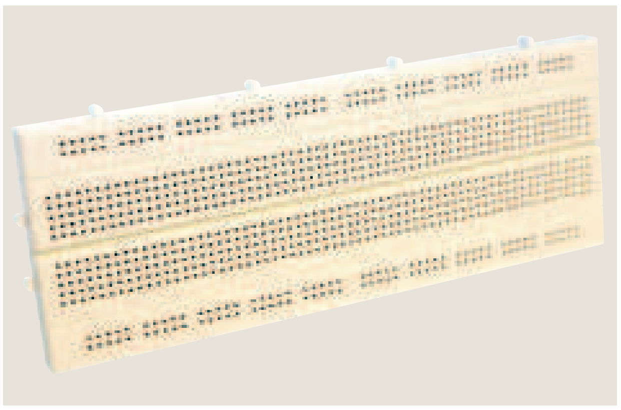
The Arduino has several built-in LEDs, such as the TX and RX LEDs that flash when data is being sent or received. We'll be using a special testing LEDs to show that the board is working OK.

Getting Arduino to work on Linux can be a little tricky, because it uses Java. What's more, it needs the official Sun Java rather than a clone. If you're using Ubuntu, make sure you have the Multiverse repository enabled, then install the packages *sun-java5-jre, gcc-avr* and *avr-libc*. Many users have trouble with braille support clashing with Arduino, so if you don't need braille support you should remove the *brltty* package first – if you plugged in your Arduino board while *brltty* was installed, unplug it and try again once *brltty* is removed. Once all the software is installed, run **sudo update-alternatives --config java** and select the number of the official Sun Java. That ought to be enough to get things working. »

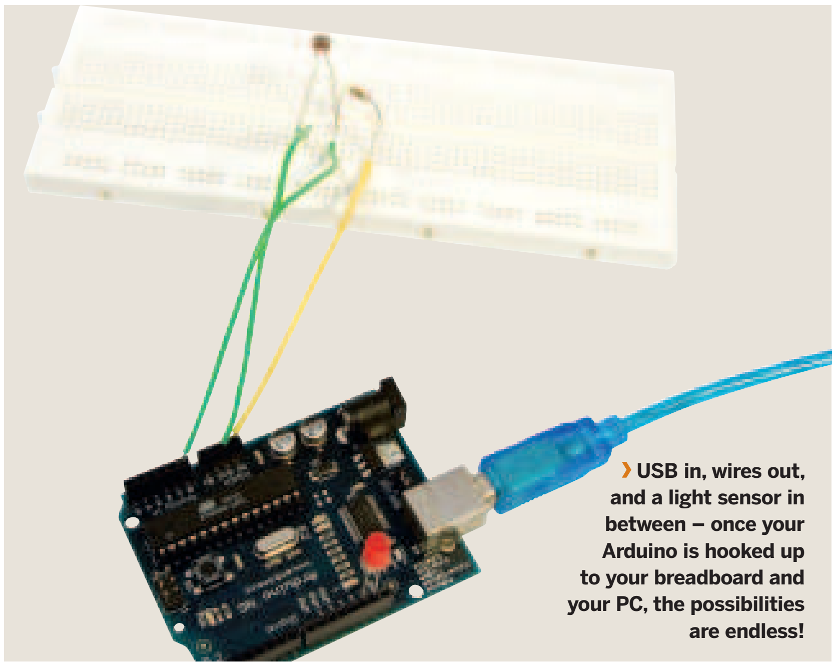


› The Arduino board is *tiny*, which means you can cram it into all sorts of cunning places.





› A full-size solderless breadboard usually is usually 60 or so pins wide, by at least 10 high. In this picture, the rows are connected and the columns are not.



› USB in, wires out, and a light sensor in between – once your Arduino is hooked up to your breadboard and your PC, the possibilities are endless!

» If you're not using Ubuntu or another Debian-based system, the same routing is required: Sun's Java, *gcc-avr* and *avr-libc*, but you probably won't have to worry about the *update-alternatives* system with other distros.

We've included the Arduino development environment (IDE) on your DVD this month – extract that to your desktop, then run the **arduino** command. If you have problems, try running it from a terminal so you can see any error messages. The first time you run the IDE, you'll be asked to choose a place where your programs should be stored – a subdirectory of your home directory is fine.

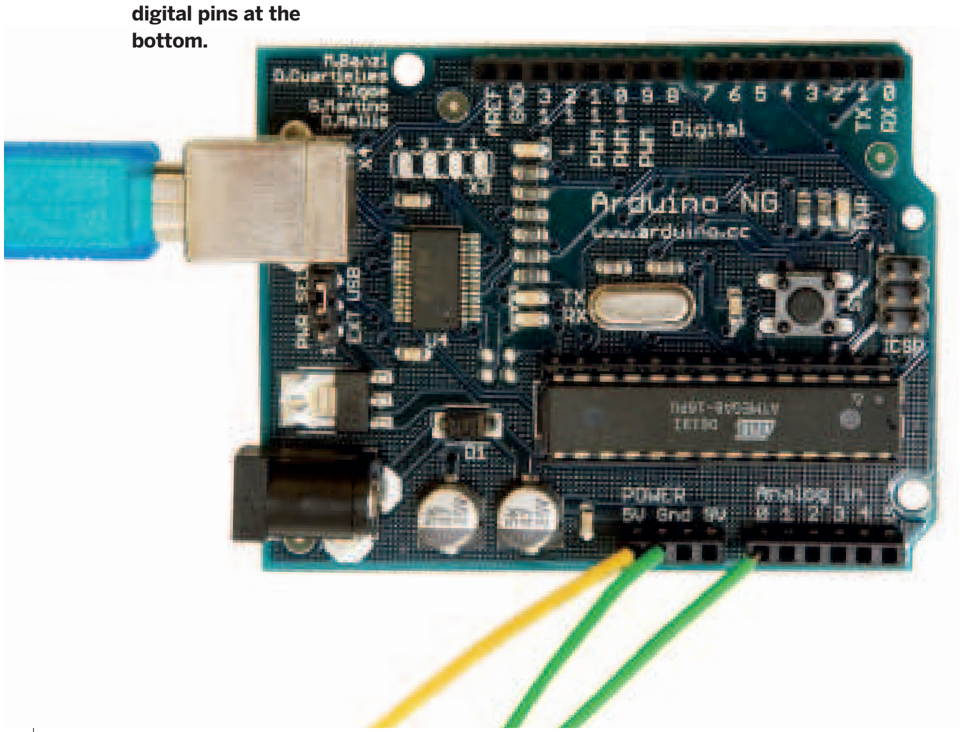
Now that you have your Arduino software installed, go ahead and hook up the Arduino to your PC using a USB cable. If the board is working OK, the PWR (power) light will be on, and your test LED might already be flashing to show that something is on the board already.

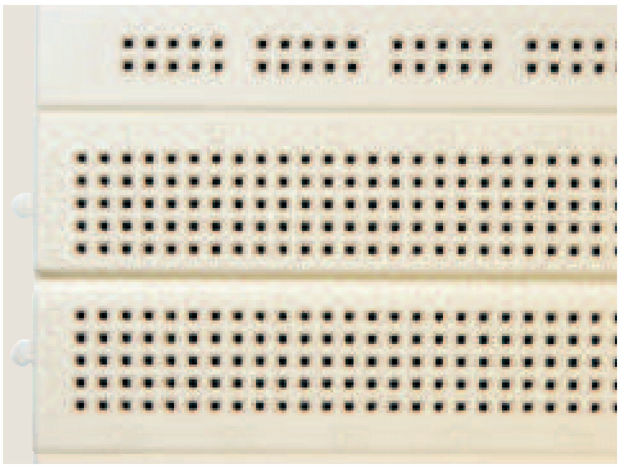
In the Arduino editor, go to File > New to enter some new code. Code files in Arduinoland are known as "sketches", and they are very easy to get started with. We're going to start with a simple project to make an LED flash on and off, and I want to show you the code and make sure it works before we look at how it works. Here's the sketch:

```
int ledPin = 13;

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

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(300);
  digitalWrite(ledPin, LOW);
  delay(100);
}
```

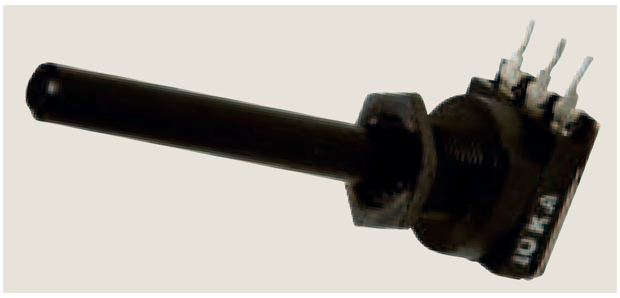
Now go to Tools > Microcontroller and choose either **atmega8** (if you have the ArduinoNG) or **atmega168** (if you have the Diecimila). If you're not sure, ignore this step – Arduino will check the CPU when it connects, and report an error if it finds the wrong CPU. If you get such an error, just choose the other option! With the CPU configured, go to Tools > Serial Port, and you'll see a list of

› The Arduino board is dominated by the ATmega chip, but you can see the analog pins at the top and the digital pins at the bottom.





› My breadboard has numbers as shown above for the rows and letters for the columns; I've referred to them throughout this feature so that you can copy my circuits exactly.



› A potentiometer reports higher analog values as you turn its knob, so make sure you use **analogRead()**!

possible USB devices. On Linux, chances are it's **/dev/ttyUSB0**, so select that.

OK, that's all the configuration done – it's time to upload your sketch to the board! On the IDE menu bar, you'll see an icon with a right-facing arrow on – that's the Upload button. If you have a Diecimila, you can click that now and your program will be uploaded. Otherwise, you need to press the little reset button on the Arduino board first so that the Arduino is ready to accept a new sketch, then press the upload button. The TX and RX lights should flicker for a second or so, followed by about five seconds of nothingness as your Arduino waits to see if any further instructions are going to arrive, and finally your little test LED should start flashing. Success!

"Should you plug your LED in the wrong way around, nothing will happen; just reverse it."

You can take this little project one step further by taking an LED from your kit bag, and plugging it into digital pin 13. If you haven't used an LED before, you should look closely at it before plugging it in: note that it has two wires coming out, that one wire is longer than the other, and that one side of the plastic-coloured rim is flat. The longer wire and the flat side are there to indicate polarity: the side with the shorter wire and the flat rim is negative. When plugging your LED into your Arduino board, make sure the positive wire goes into pin 13, and the negative wire goes into GND (Ground), and all being well you should seeing it flash in time with the test LED. If you get it the wrong way around, nothing will happen, but don't leave the LED plugged in like that too long!

How the code works

Now that your Arduino board works properly, I want to explain how the code works so you can try modifying it yourself:

» **int ledPin = 13** defines a variable, **ledPin**, that holds an integer (a whole number, *ie* not numbers like 3.1) and is giving the starting value of 13.

» **setup()** is a default function for the Arduino. It gets called when your program starts so you can set up some basic configuration.

» **pinMode(ledPin, OUTPUT)** tells the Arduino that you want to send data to pin number 13 rather than read data.

» **loop()** is another default function for the Arduino, and is called every time the CPU is looking for some work to do.

» **digitalWrite(ledPin, HIGH)** means "send the value HIGH to pin 13" – **HIGH** is equivalent to a binary 1 versus a binary 0, or an "on" compared to an "off". In this case, it turns the LED on.

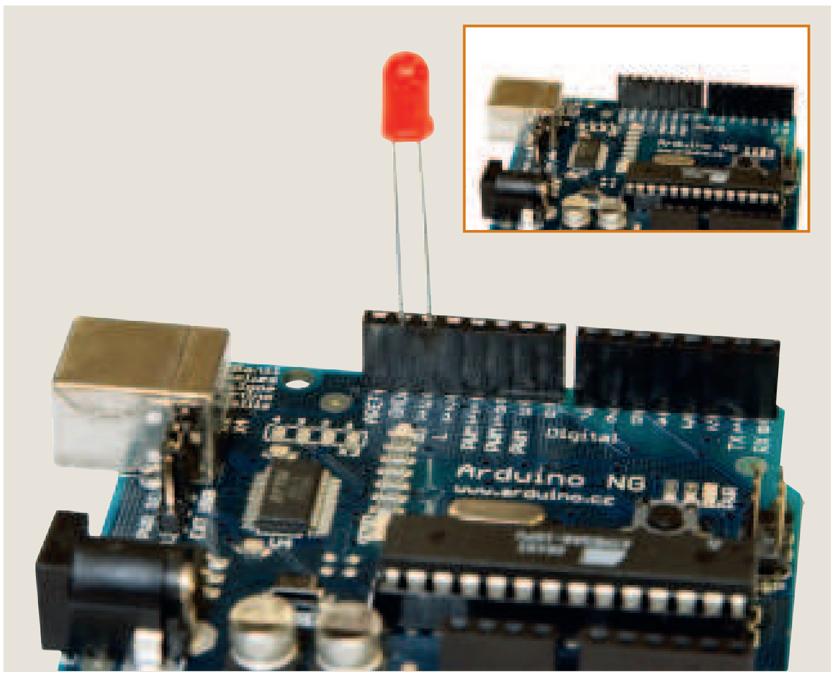
» **delay(300)** makes the CPU pause for 300 milliseconds, or about a third of a second.

» **digitalWrite(ledPin, LOW)** turns the LED off.

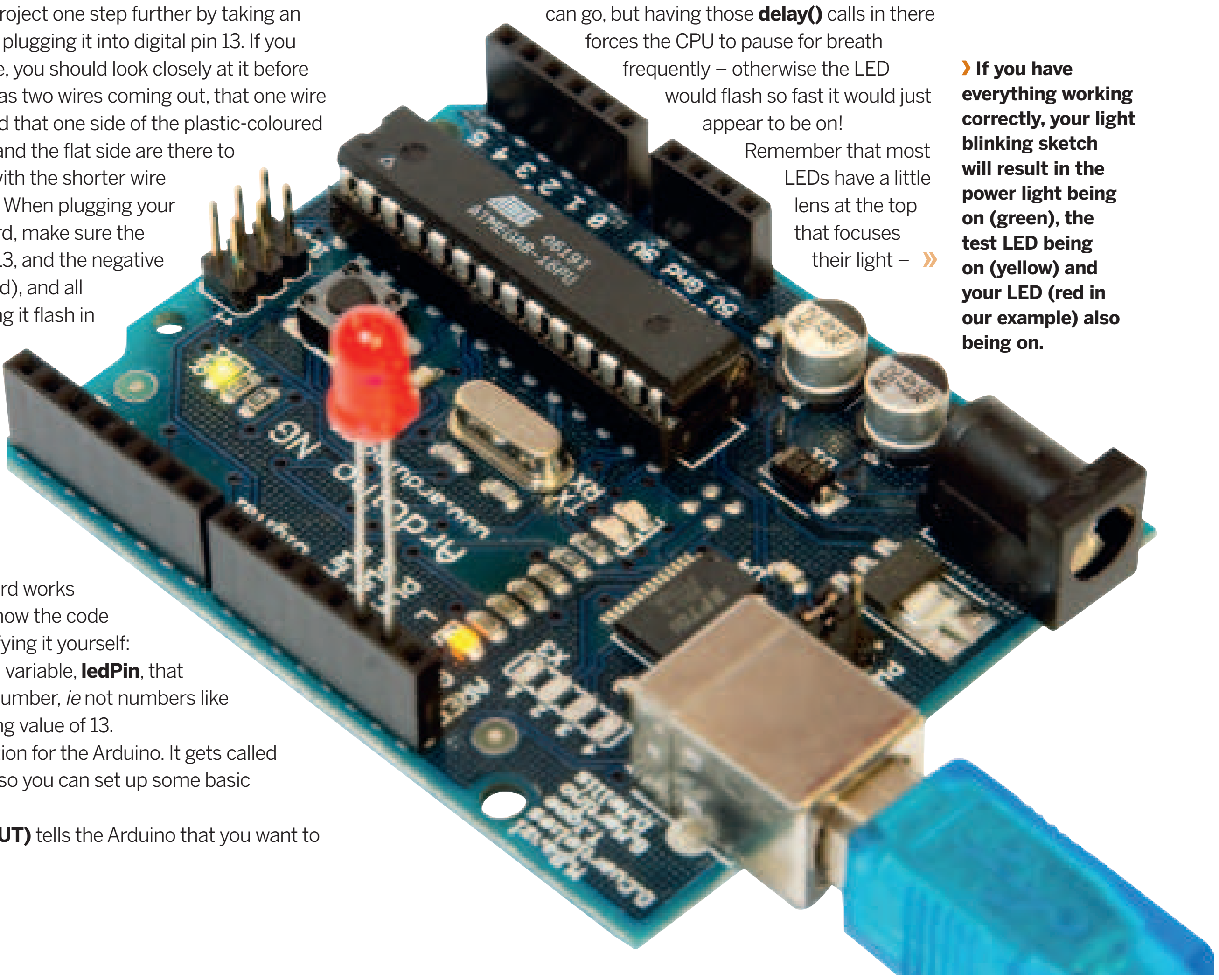
» delay(100) makes the CPU wait a tenth of a second

And that's it! The **loop()** function gets called as fast as the CPU can go, but having those **delay()** calls in there forces the CPU to pause for breath frequently – otherwise the LED would flash so fast it would just appear to be on!

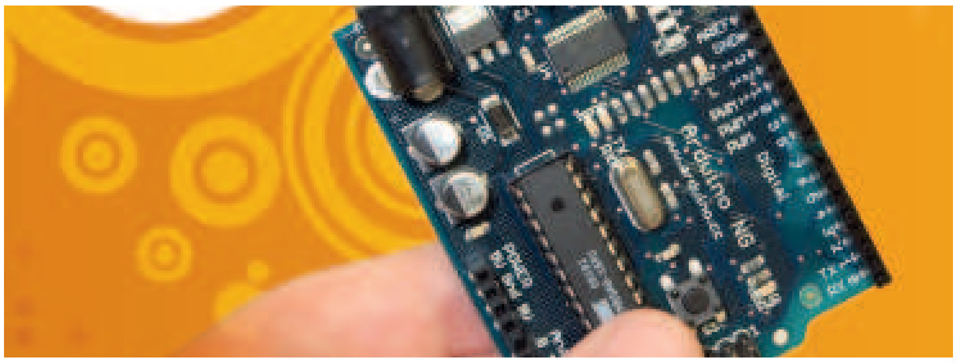
Remember that most LEDs have a little lens at the top that focuses their light – »

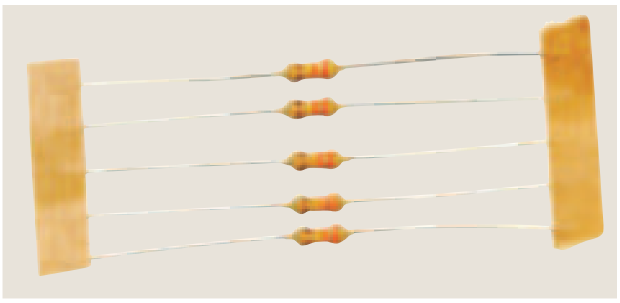


› Connect the LED digital pin 13/GND, but make sure you look at the wire length on the LED to see which side is positive!

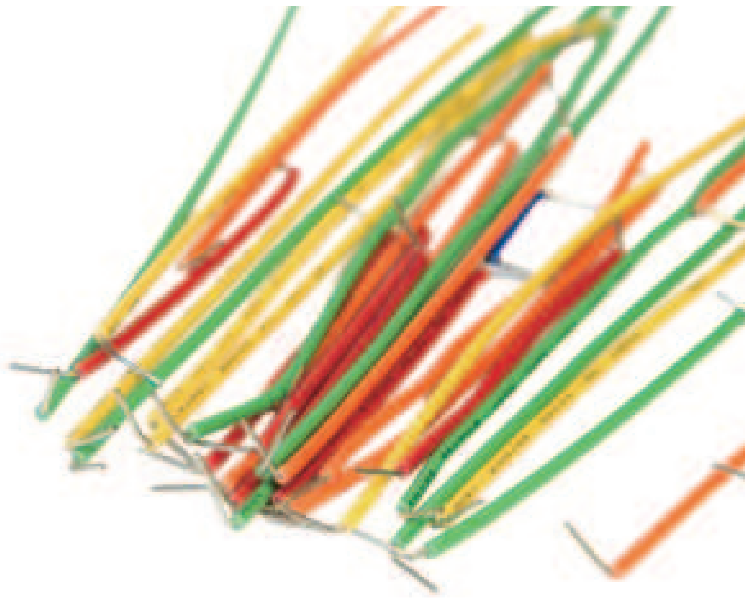


› If you have everything working correctly, your light blinking sketch will result in the power light being on (green), the test LED being on (yellow) and your LED (red in our example) also being on.





› Resistors are so cheap you usually buy at least five at a time – make sure and look closely at the stripes to determine how much resistance they offer.



› You can buy a bag of pre-cut wires from any good electrical store. If you want to look really smart, you can even use colours for certain types of connections!

» you'll probably find your LED appears brightest when you look straight down at it.

Using the breadboard

The solderless breadboard you have has strips of metal hidden away under the holes, connecting rows of pins together as a circuit. The columns aren't connected, which means they don't carry a connection horizontally – only vertically. If you have a full-size breadboard like ours, you'll have a gap between rows; the rows on either side of these gaps aren't connected either. To give you an example of how this works, we can modify the simple LED system we have now by introducing the breadboard and two wires. My board has columns numbered 1 to 60 and rows lettered A to J, as is shown in the photos of it on the previous page, and I'll be using these numbers to explain where to put your wiring. Even if yours doesn't have similar numbering, these numbers will still be helpful for you because they tell you what row and column I'm using, and that's all that really matters.

"Refer to your resistor 'cheat sheet' so that you can decipher what their coloured stripes mean."

So, to move the blinkenlights "project" over to the breadboard, connect a wire from GND to J33, and another wire from Digital 13 to J32. Now, that has hooked up current to I33 and I32, H33 and H32, G33 and G32, and F33 and F32. My breadboard jump is between row F and row E, which means E33 and E32 aren't part of the circuit. Now all you have to do is put your LED in the right place to finish the circuit: where do you think it should go?

Unless you're very fearful of electric circuits, you can probably just have a dabble and try things until the LED lights up – you learn a lot by dabbling, and it's more fun, too! In this example, plugging your LED into F33 (negative; short wire) and F32 (position; long wire) would make it light up, but so would plugging into column G, H or I.

Now, let's try something else: over on the other side of your Arduino board are Analog In pins, as well as a few marked Power. These carry a lot more current – note than one is marked 5V and one is marked 9V? That's where we can get more power from, but you need to be careful – if you put too much through an LED, you can burn it out. That's not a big problem if you bought a mountain of LEDs, but if you only have three or four you need to make them last! However, it's safe to plug it in for a second or two just to make sure it works, so please do that now: unplug your Arduino so it has no power, then connect 5V to J33 and the Gnd next to 5V to J32. Now plug it in for a second: your LED should light up much, more brightly (and may even start to get warm!). Once you can see that it works, unplug the USB cable again so the light goes back out.

Now, let's up the complexity a little: I want to introduce you to resistors. These little things produce a voltage drop of different levels, depending on the variety of resistor you have. The exact resistance is shown in the little coloured stripes on the resistors, but these are absolutely meaningless for mere mortals – don't be afraid to keep around a resistor cheat sheet showing what the different stripes mean. For now, connect 5V to J38 so that the LED no longer has power going to it. To complete the circuit, you need to connect a resistor from I38 to J33 , and now you should see the LED light – just a bit dimmer. If the resistor colour codes are meaningless to you, try using different resistors here and arranging them by strength based on the LED brightness!

Reading input

In your kit bag you should have a light sensor, which reports different values depending on the amount of light it receives. This takes a few more wires, an extra function call, as well as a conditional statement, but it's really not so hard – and it's worth the effort because you can start writing code to make the Arduino aware of its environment! I have two special lanes on my breadboard designed to make it easy to conduct power, but I won't be using that here because smaller breadboards don't always come with something similar.

First, plug your light sensor into your breadboard. I've put mine on F34 and F37. Plug a strong resistor into G34, and connect it to I29, then connect a wire from J29 to 5 – this gives your light sensor power. You also need to connect a wire from G37 to the Gnd pin next to 5V, so that the circuit is complete. To read a value from the light sensor, connect another wire from H34 to Analog 0, and that's your system ready to be programmed!

Reading from an analog input in an Arduino sketch is very easy: you need to store which pin you're reading from (0), as well as store the value from the light sensor. This is all done using the **analogRead()** function, which returns a value you can store and work with. In a sketch it looks like this:

```
int ledPin = 13;
int lgtPin = 0;
int lgtVal = 0;

void setup()
```

Hardware in LXF

We very rarely cover hardware like the Arduino in *Linux Format* – in fact, this is a bit of a first for us! So, please write in and let us know what you think: do you want to see us write some follow ups to this primer, using more parts, more complex sketches, and perhaps even some complete projects? Or would you rather we stayed away from hardware altogether? Let us know! **linuxformat@futurenet.co.uk**

"Reading from an analog input in an Arduino sketch is very easy – use the analogueRead() function."

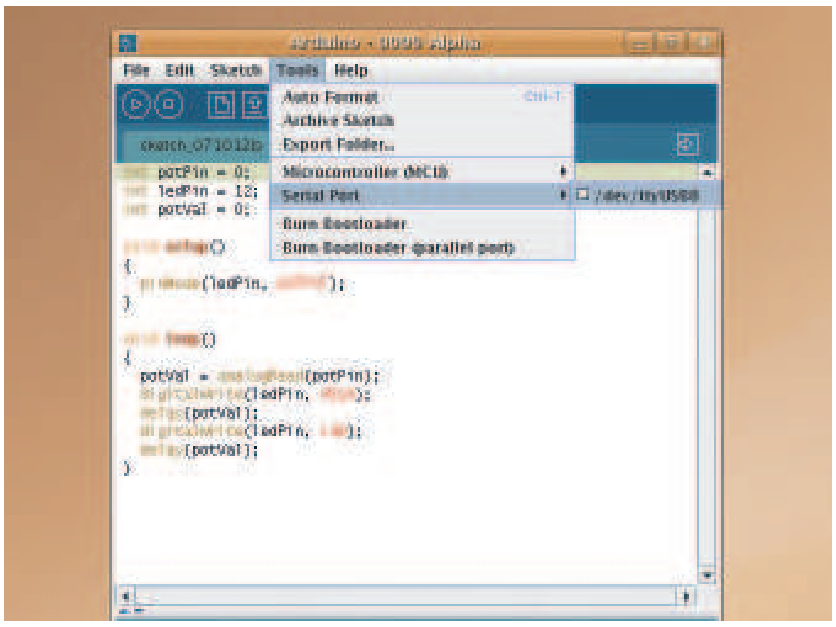
```
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  lgtVal = analogRead(lgtPin);
  digitalWrite(ledPin, HIGH);
  delay(lgtVal);
  digitalWrite(ledPin, LOW);
  delay(lgtVal);
}
```

What do you think that sketch will do? If you're not sure, try assembling it, uploading the sketch to your board, and covering/revealing the light sensor!

More to try...

If you managed to pick up a potentiometer, these are particularly easy to drop in once you have your light sensor working: your cable to Analog 0 needs to connect to the middle pin of the potentiometer; your cable to Ground needs to connect to the left pin; and your cable to the power (via a resistor) should connect to the right pin. Make sure your potentiometer is very firmly plugged into the breadboard – they usually have thicker connectors, so you might have to give it a good push!

Of course, this has only really been a very light overview of what Arduino can do, and that's because the real magic comes in the software you build. We've shown you here how to put together circuits to read and write data, and the next question is: what are you going to do with it? LXF



```
int potPin = 0;
int ledPin = 12;
int potVal = 0;

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

void loop()
{
  potVal = analogRead(potPin);
  digitalWrite(ledPin, HIGH);
  delay(potVal);
  digitalWrite(ledPin, LOW);
  delay(potVal);
}
```

› Before you try uploading to a board, make sure you select the correct serial port for your Arduino – it's probably /dev/ttyUSB0 or something similar.