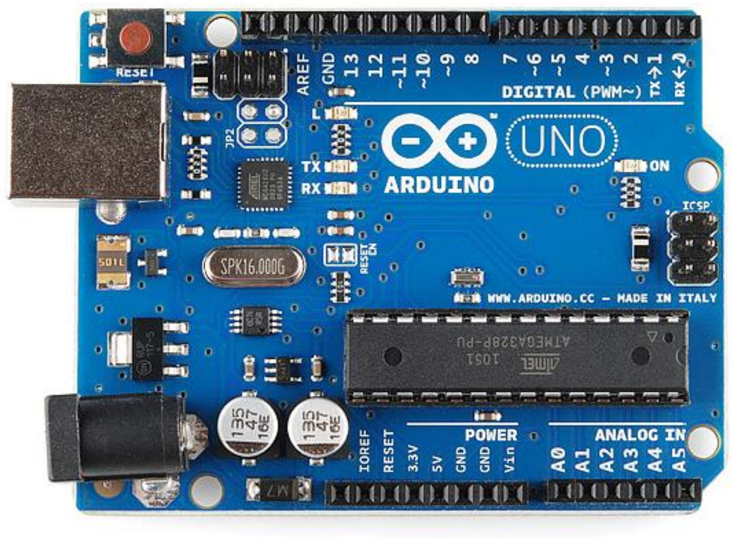
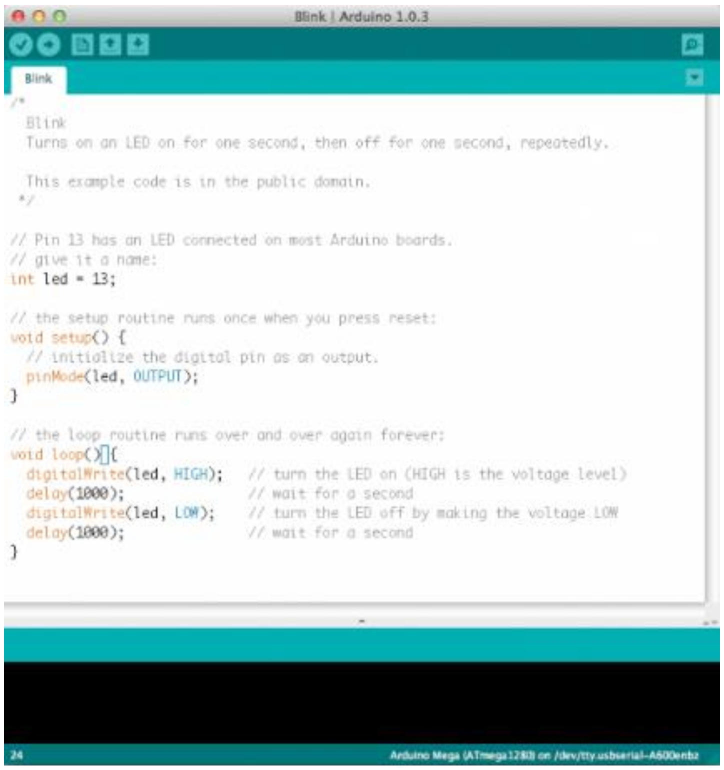
# Introduction

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.





```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

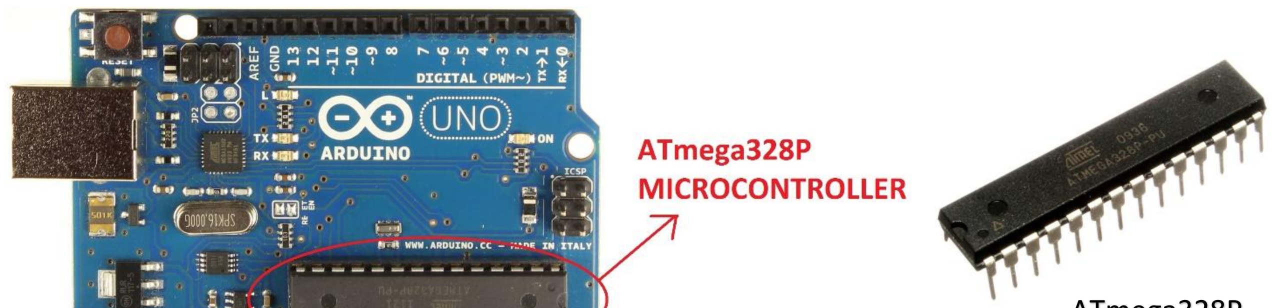
```
24                    Arduino Mega (ATmega1280) on /dev/tty.usbserial-A600enbz
```

What Does it Do?

The Arduino hardware and software was designed for artists, designers, hobbyists, hackers, newbies, and anyone interested in creating interactive objects or environments. Arduino can interact with buttons, LEDs, motors, speakers, GPS units, cameras, the internet, and even your smart-phone! This flexibility combined with the fact that the Arduino software is free, the hardware boards are pretty cheap, and both the software and hardware are easy to learn has led to a large community of users who have contributed code and released instructions for a **huge** variety of Arduino-based projects.

What is a microcontroller?

A microcontroller is an integrated computer on a chip. The microcontroller is the heart (or, more appropriately, the brain) of the Arduino board. The Arduino development board is based on AVR microcontrollers of different types, each of which have different functions and features.



ATmega328P

Microcontroller chip

There are hundreds different microcontroller families from many different manufactures. However, there are two family of microcontroller that are extremely popular among hobbyist. The **“PIC”** series from **<u>Microchip</u>** and **“AVR”** series from **<u>Atmel</u>**. Both these chips are wonder of modern microelectronics. PIC had ruled for a long time but now AVR is also getting in serious competition. Arduinos also use AVR family of microcontrollers.

Speed and ease of use.

I prefer AVRs because one major reason. They are fast. When a PIC and an AVR is running with same frequency let’s say 16 MHz, then the AVR is actually executing four times faster than the PIC ! Yes 4 times faster. This is because the PIC requires 4 cycle to perform a single execute cycle while the AVR execute most of the instruction in 1 clock cycle. In addition, I like the AVR architecture because of its consistency. It makes using the most advanced feature of AVR very easy to use. These chips are easily available and they are cheap.

Free “C” compiler.

One more important thing, generally microcontroller programs are written in assembly language for efficiency. Which you may know is a very low level and unstructured language. Therefore, to achieve a small thing lots of code need to be written and the programmer cannot concentrate on program logic. This makes the things harder. However, mid to high end AVR MCUs are powerful enough to support high level language such as C efficiently. To write programs in ‘c’ for AVR or any other microcontroller we need a c compiler for that MCU. Generally these compilers are priced so high that they are out of reach of hobbyist and small companies. But fortunately for AVR MCUs there is a very high quality 'c' compiler for free. It is GNU C compiler. It is extremely popular and it has a large user base.

Different AVRs

Few Common Members of AVR Family

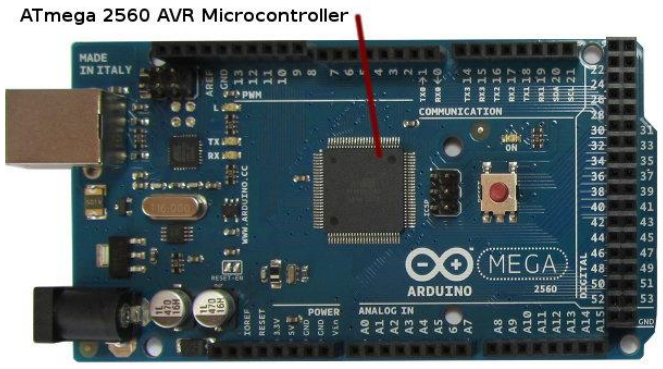
Now we have selected AVR as our choice of microcontroller, let us see what different AVRs are available. The popular microcontrollers of AVR family are

- AT TINY2313 [20 PIN, 2K Flash ,128 Byte RAM, 128 Bytes EEPROM]
- **ATmega8** [28 PIN , 8K Flash , 1KB Ram, 512 Byte EEPROM]
- ATmega16 [ 40 PIN, 16K Flash, 1KB Ram, 512 Byte EEPROM]
- **ATmega32** [ 40 PIN, 32K Flash, 2KB Ram, 1KB EEPROM]

You can choose any one of them according to your need. In the above list we have mentioned the key parameter of the MCU. These parameters includes:-

- **The number of pins** that MCU is having for example ATmeg8 has 28 pins while ATmega16 has 40 pins. The more the number of pin a MCU has the more external hardware you can add to it at the same time. But it will be bigger in size and costly. So for cheaper and smaller final product you need to select the MCU with the minimum number of pins you can manage to attach all external peripherals.
- **Amount of Flash Memory:** The flash memory is where the main program of MCU is stored. Thus more the flash memory the bigger and more complex program it can store. In the same time a MCU with larger flash memory tends to be more expensive. So the bottom line is to select the MCU which can just hold your final program. Example ATmega8 has 8K Flash.
- **Amount of RAM:** All the data hold by the MCU during runtime are stored in RAM. So more the ram it can store more data at runtime. ATmega8 has 1KB RAM.

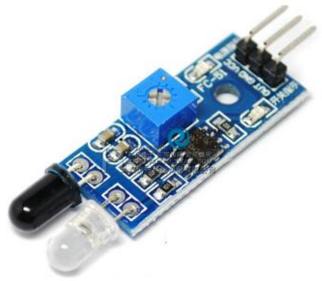
- **Amount of EEPROM:** If the MCU program collected some data (like a password from user) it is stored in RAM. But MCU cannot retain this data if the power to MCU is interrupted. To hold data without the power supply MCU need to write it to its internal EEPROM. Once data is written to EEPROM. It does NOT require power to retain its value. On next startup the MCU can read this data from EEPROM. From the list you can see that ATmega8 has 512 bytes of EEPROM while ATmega32 has 1KB of EEPROM.



**Inputs**

Some examples of inputs would be a temperature sensor, a motion sensor, a distance sensor, a switch and so forth.

Distance Sensor



IR Sensor

Temperature and Humidity Sensor

**Outputs**

Some examples of outputs would be a light, a screen, a motor and so forth.

DC Motor

LED lights

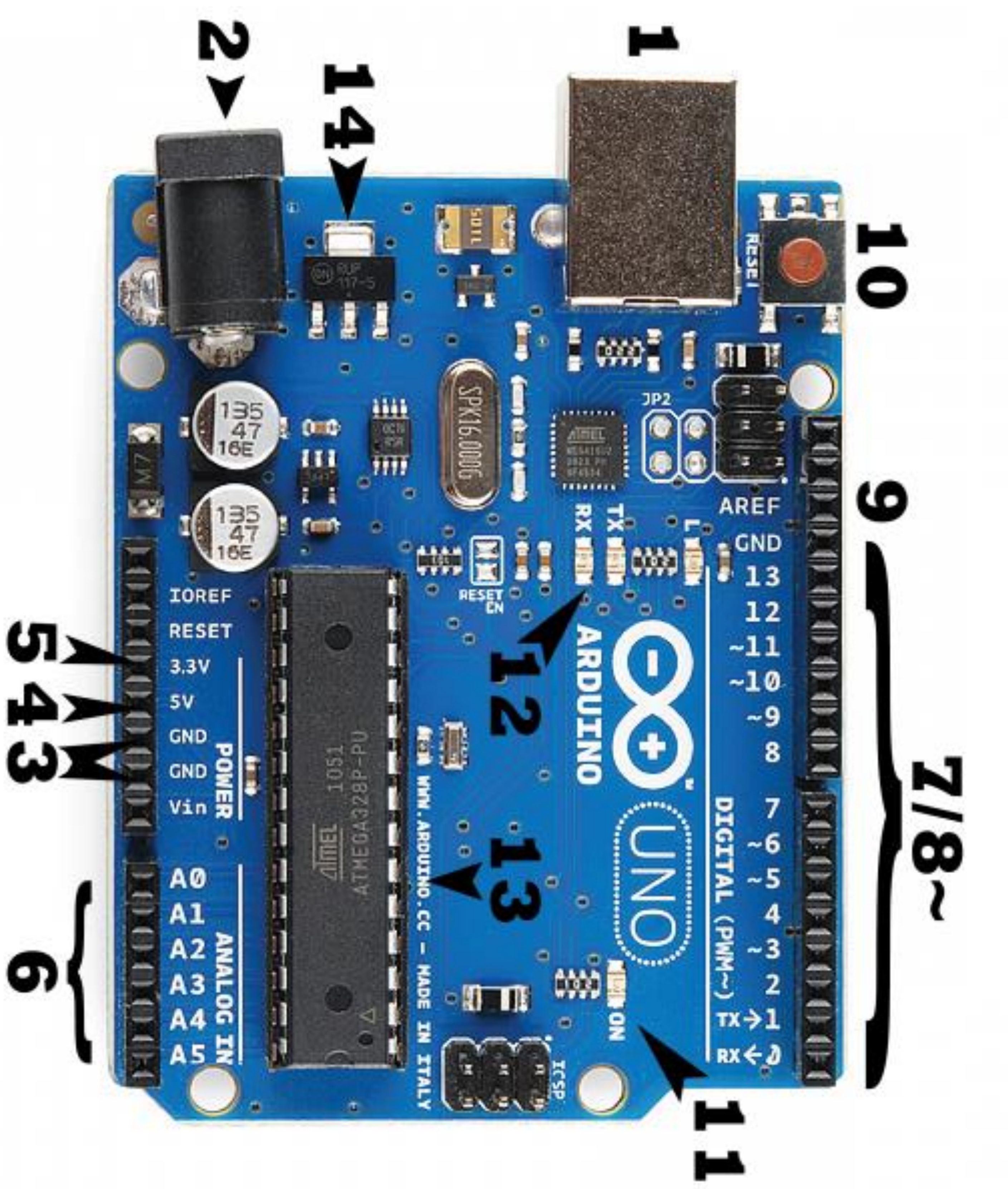
LCD Screen

What's on the board?

There are many varieties of Arduino boards that can be used for different purposes. Some boards look a bit different from the one below, but most Arduinos have the majority of these components in common:

Power (USB / Barrel Jack)

Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from your computer or a wall power supply that is terminated in a barrel jack. In the picture above the USB connection is labeled **(1)** and the barrel jack is labeled **(2)**.


1
2
14
10
9
7/8~
12
13
5
4
3
6
11
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
ARDUINO
UNO
RX
TX
L
ON
RESET EN
ICSP
JP2
WWW.ARDUINO.CC – MADE IN ITALY
ATMEGA328P-PU

The USB connection is also how you will load code onto your Arduino board.

**NOTE:** Do NOT use a power supply greater than 20 Volts as you will overpower (and thereby destroy) your Arduino. The recommended voltage for most Arduino models is between 6 and 12 Volts.

## Pins (5V, 3.3V, GND, Analog, Digital, PWM, AREF)

The pins on your Arduino are the places where you connect wires to construct a circuit (probably in conjuction with a breadboard and some wire. They usually have black plastic 'headers' that allow you to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

- **GND (3)**: Short for 'Ground'. There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- **5V (4) & 3.3V (5)**: As you might guess, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.
- **Analog (6)**: The area of pins under the 'Analog In' label (A0 through A5 on the UNO) are Analog In pins. These pins can read the signal from an analog sensor (like a temperature sensor) and convert it into a digital value that we can read.
- **Digital (7)**: Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED).
- **PWM (8)**: You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM). These pins are able to simulate analog output (like fading an LED in and out).
- **AREF (9)**: Stands for Analog Reference. Most of the time you can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

## Reset Button

Arduino has a reset button **(10)**. Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if your code doesn't repeat, but you want to test it multiple times.

## Power LED Indicator

Just beneath and to the right of the word "UNO" on your circuit board, there's a tiny LED next to the word 'ON' **(11)**. This LED should light up whenever you plug your Arduino into a power source. If this light doesn't turn on, there's a good chance something is wrong. Time to re-check your circuit!

## TX RX LEDs

TX is short for transmit, RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for serial communication. In our case, there are two places on the Arduino UNO where TX and RX appear – once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs **(12)**. These LEDs will give us some nice visual indications whenever our Arduino is receiving or transmitting data (like when we're loading a new program onto the board).

## Main IC

The black thing with all the metal legs is an IC, or Integrated Circuit **(13)**. Think of it as the brains of our Arduino. The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC's from the ATMEL company. This can be important, as you may need to know the IC type (along with your board type) before loading up a new program from the Arduino software. This information can usually be found in writing on the top side of the IC. If you want to know more about the difference between various IC's, reading the datasheets is often a good idea.

## Voltage Regulator

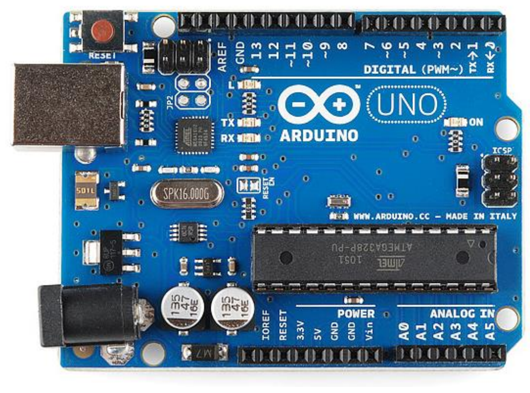
The voltage regulator **(14)** is not actually something you can (or should) interact with on the Arduino. But it is potentially useful to know that it is there and what it's for. The voltage regulator does exactly what it says – it controls the amount of voltage that is let into the Arduino board. Think of it as a kind of gatekeeper; it will turn away an extra voltage that might harm the circuit. Of course, it has its limits, so don't hook up your Arduino to anything greater than 20 volts.

## The Arduino Family

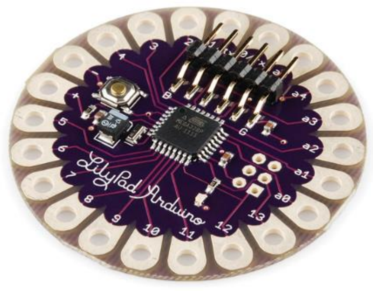
Arduino makes several different boards, each with different capabilities. In addition, part of being open source hardware means that others can modify and produce derivatives of Arduino boards that provide even more form factors and functionality. If you're not sure which one is right for your project, check this guide for some helpful hints. Here are a few options that are well-suited to someone new to the world of Arduino:

### Arduino Uno (R3)

The Uno is a great choice for your first Arduino. It's got everything you need to get started, and nothing you don't. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a USB connection, a power jack, a reset button and more. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.



LilyPad Arduino

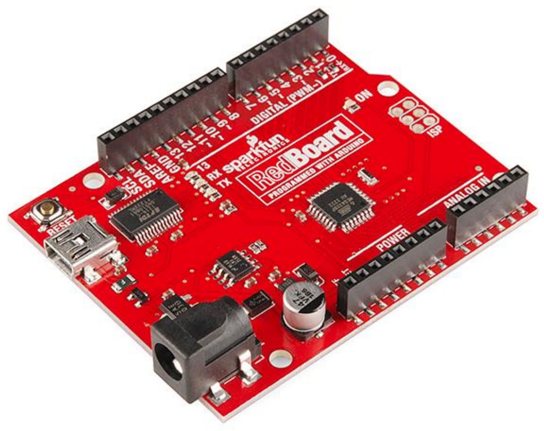


This is LilyPad Arduino main board! LilyPad is a wearable e-textile technology developed by Leah Buechley and cooperatively designed by Leah and SparkFun. Each LilyPad was creatively designed with large connecting pads and a flat back to allow them to be sewn into clothing with conductive thread. The LilyPad also has its own family of input, output, power, and sensor boards that are also built specifically for e-textiles. They're even washable!

RedBoard

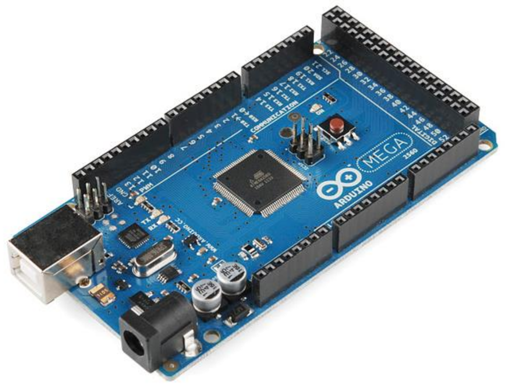
At SparkFun we use many Arduinos and we're always looking for the simplest, most stable one. Each board is a bit different and no one board has everything we want – so we decided to make our own version that combines all our favorite features.

The RedBoard can be programmed over a USB Mini-B cable using the Arduino IDE. It'll work on Windows 8 without having to change your security settings (we used signed drivers, unlike the UNO). It's more stable due to the USB/FTDI chip we used, plus it's completely flat on the back, making it easier to embed in your projects. Just plug in the board, select "Arduino UNO" from the board menu and you're ready to upload code. You can power the RedBoard over USB or through the barrel jack. The on-board power regulator can handle anything from 7 to 15VDC.



<u>Arduino Mega (R3)</u>

The Arduino Mega is like the UNO's big brother. It has lots (*54!*) of digital input/output pins (14 can be used as PWM outputs), 16 analog inputs, a USB connection, a power jack, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The large number of pins make this board very handy for projects that require a bunch of digital inputs or outputs (like lots of LEDs or buttons).



<u>Arduino Leonardo</u>

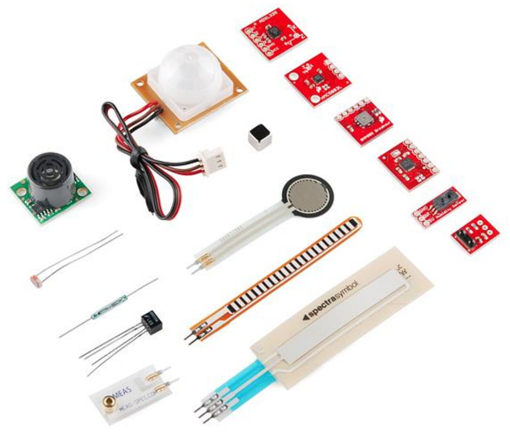
The Leonardo is Arduino's first development board to use one microcontroller with built-in USB. This means that it can be cheaper and simpler. Also, because the board is handling USB directly, code libraries are available which allow the board to emulate a computer keyboard, mouse, and more!

## The Extended Family

While your Arduino board sure is pretty, it can't do a whole lot on its own – you've got to hook it up to something. There are lots of tutorials here on learn as well as the links back in the 'What does it do' section, but rarely do we talk about the general *kinds* of things you can easily hook into. In this section we'll introduce basic **sensors** as well as Arduino **shields**, two of the most handy tools to use in bringing your projects to life.

## Sensors

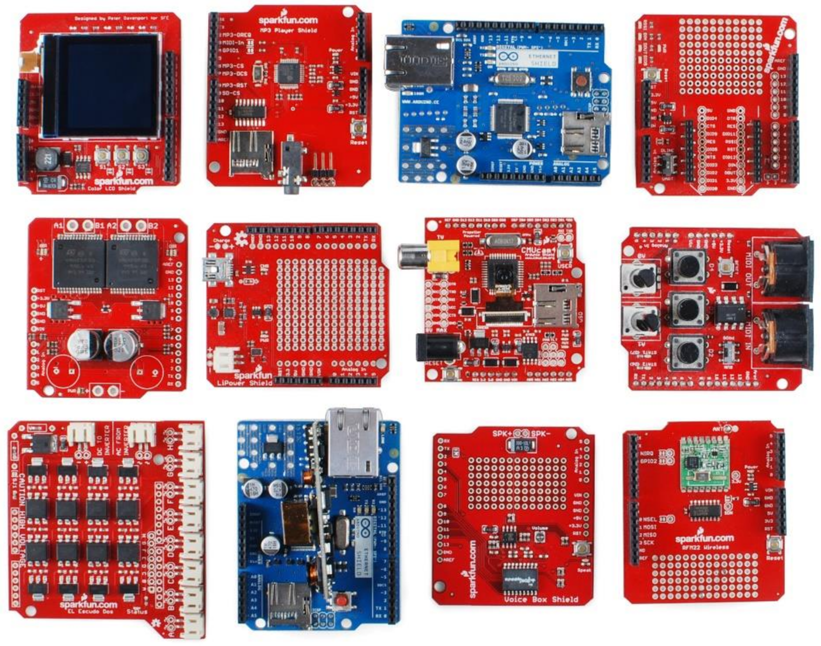
With some simple code, the Arduino can control and interact with a wide variety of **sensors** - things that can measure light, temperature, pressure , acceleration, carbon monoxide, humidity, barometric pressure, you name it, you can sense it!



*Just a few of the sensors that are easily compatible with Arduino*

## Shields

Additionally, there are these things called **shields** – basically they are pre-built circuit boards that fit on top of your Arduino and provide additional capabilities – controlling motors, connecting to the internet, controlling an LCD screen, and much more.



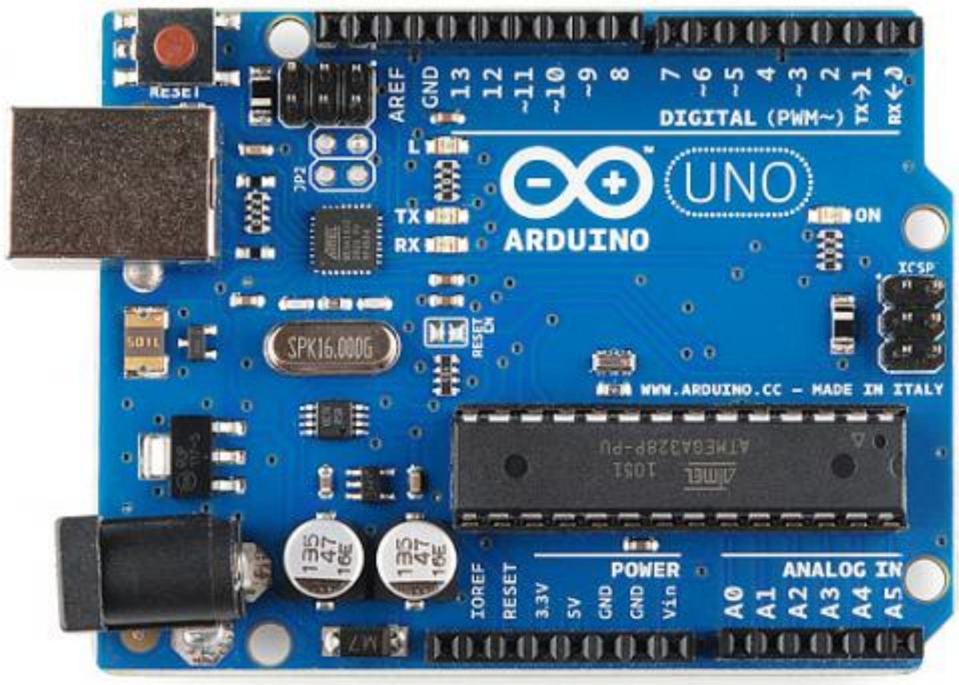
*A partial selection of available shields to extend the power of your Arduino*

## Installing Arduino IDE

What you will need:

- A computer (Windows, Mac, or Linux)
- An Arduino-compatible microcontroller

- A USB A-to-B cable, or another appropriate way to connect your Arduino-compatible microcontroller to your computer



*An Arduino Uno*

*An A-to-B USB Cable*

## Windows

This page will show you how to install and test the Arduino software with a Windows operating system (Windows 8, Windows 7, Vista, and XP).

### Windows 8, 7, Vista, and XP

- Go to the Arduino download page and download the latest version of the Arduino software for Windows.
- When the download is finished, un-zip it and open up the Arduino folder to confirm that yes, there are indeed some files and sub-folders inside. The file structure is important so don't be moving any files around unless you really know what you're doing.
- Power up your Arduino by connecting your Arduino board to your computer with a USB cable (or FTDI connector if you're using an Arduino pro). You should see the an LED labed 'ON' light up. If you're running Windows 8, you'll need to disable driver signing, so go see the Windows 8 section. If you're running Windows 7, Vista, or XP, you'll need to install some drivers, so head to the Windows 7, Vista, and XP section down below.

### Windows 8

**Caution:** Disabling your operating system's device signature verification can put your operating system at risk. You should only install drivers that you trust!

Windows 8 comes with a nice little security 'feature' that 'protects' you from unsigned driver installation. Some older versions of Arduino Uno come with unsigned drivers, so in order to use your Uno, you'll have to tell Windows to disable driver signing. This issue has been addressed in newer releases of the Arduino IDE, but if you run into issues, you can try this fix first.

To *temporarily* disable driver signing:

- From the Metro Start Screen, open Settings (move your mouse to the bottom-right-corner of the screen and wait for the pop-out bar to appear, then click the Gear icon)
- Click 'More PC Settings'
- Click 'General'
- Scroll down, and click 'Restart now' under 'Advanced startup'.
- Wait a bit.
- Click 'Troubleshoot'.
- Click 'Advanced Options'
- Click 'Windows Startup Settings'
- Click Restart.
- When your computer restarts, select 'Disable driver signature enforcement' from the list.

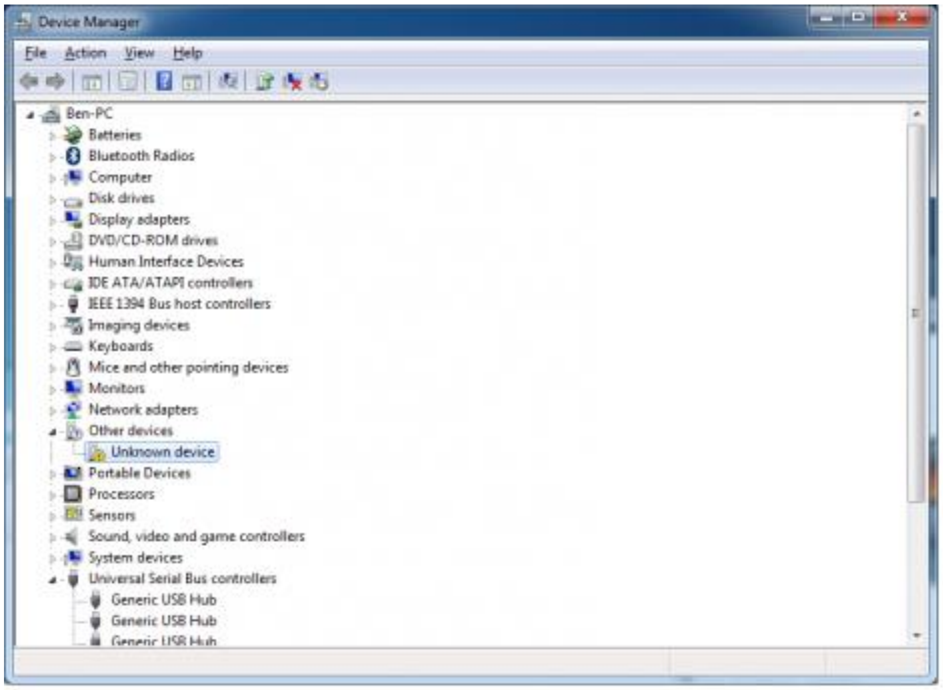
To *permanently* disable driver signing (recommended, but has some minor security implications):

- Go to the metro start screen
- Type in “cmd”
- Right click “Command Prompt” and select “Run as Administrator” from the buttons on the bottom of your screen
- Type/paste in the following commands: bcdedit -set loadoptions DISABLE_INTEGRITY_CHECKS bcdedit -set TESTSIGNING ON
- Reboot!

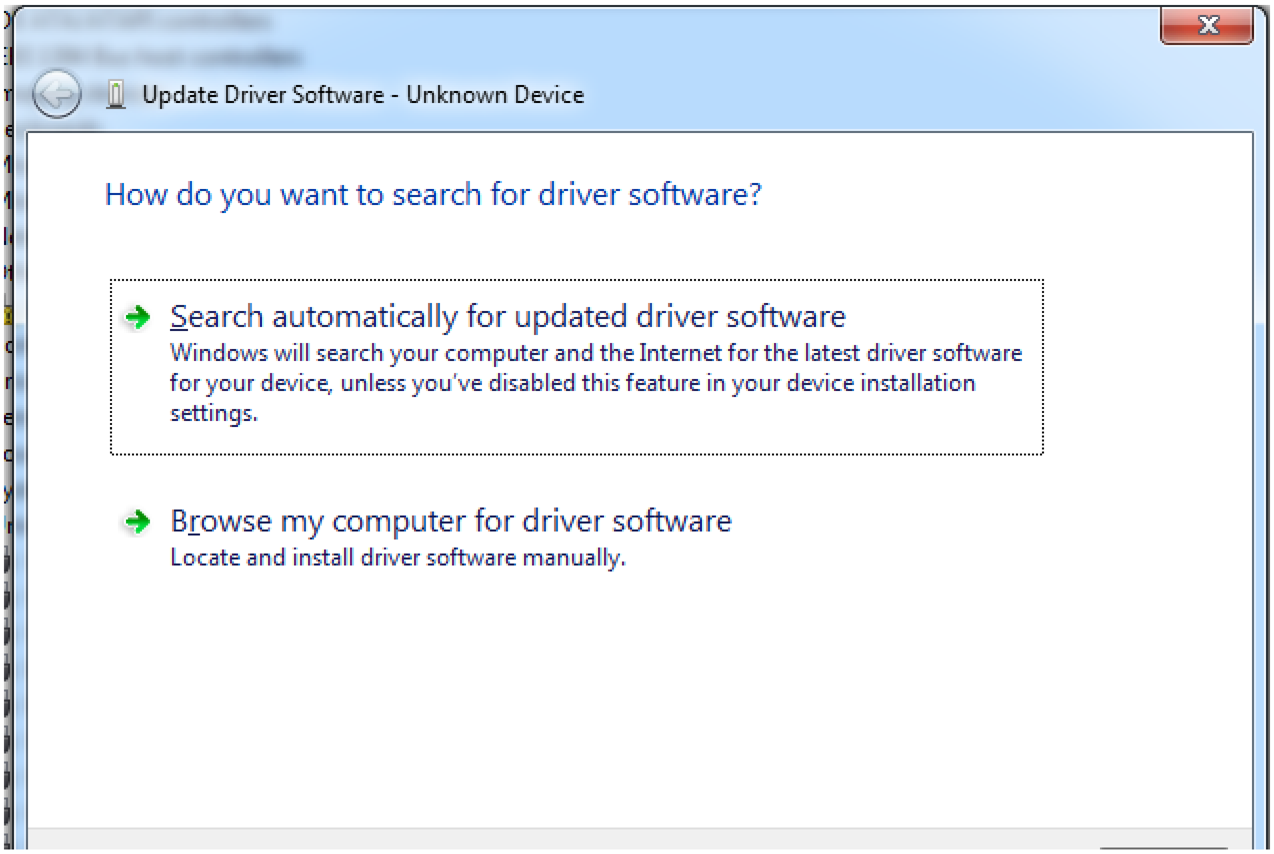
Windows 7, Vista, and XP

Installing the Drivers for the Arduino Uno (from Arduino.cc)

- Plug in your board and wait for Windows to begin it’s driver installation process
- After a few moments, the process will fail, despite its best efforts
- Click on the Start Menu, and open up the Control Panel
- While in the Control Panel, navigate to System and Security. Next, click on System
- Once the System window is up, open the Device Manager
- Look under Ports (COM & LPT). You should see an open port named “Arduino UNO (COMxx)”. If there is no COM & LPT section, look under ‘Other Devices’ for ‘Unknown Device’



- Right click on the “Arduino UNO (COMxx)” or “Unknown Device” port and choose the “Update Driver Software” option
- Next, choose the “Browse my computer for Driver software” option



Update Driver Software - Unknown Device

How do you want to search for driver software?

Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

Browse my computer for driver software
Locate and install driver software manually.

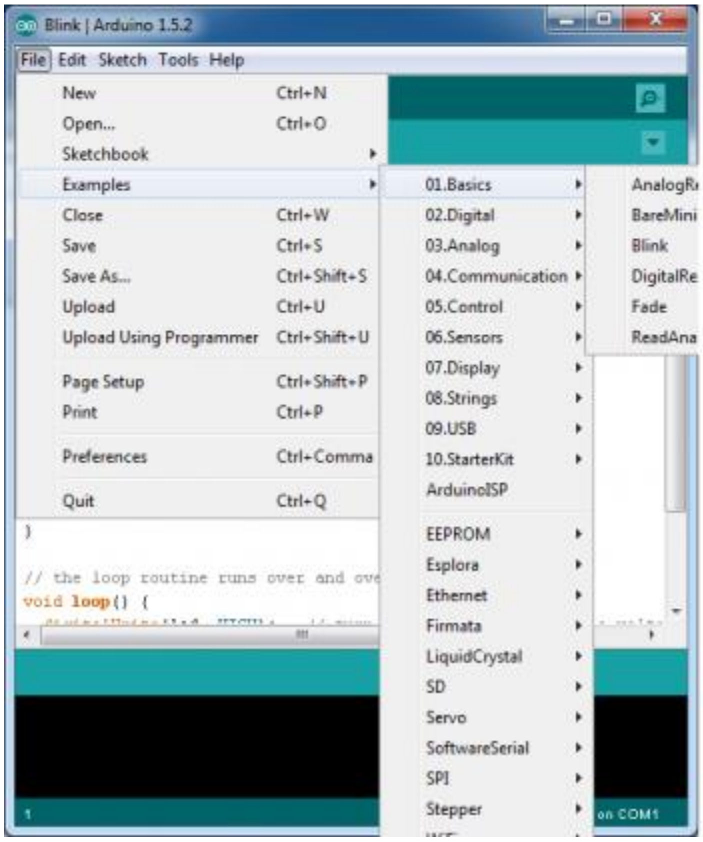
- Finally, navigate to and select the Uno’s driver file, named “ArduinoUNO.inf”, located in the “Drivers” folder of the Arduino Software download (not the “FTDI USB Drivers” sub-directory). If you cannot see the .inf file, it is probably just hidden. You can select the ‘drivers’ folder with the ‘search sub-folders’ option selected instead.
- Windows will finish up the driver installation from there

For earlier versions of the Arduino boards (e.g.Arduino Duemilanove, Nano, or Diecimila)

Launch and Blink!

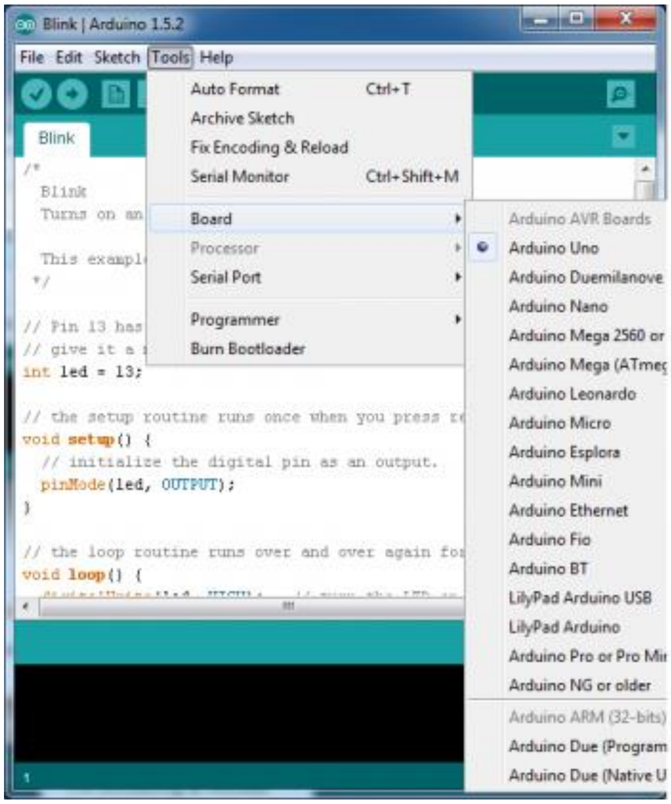
After following the appropriate steps for your software install, we are now ready to test your first program with your Arduino board!

- Launch the Arduino application
- If you disconnected your board, plug it back in
- Open the Blink example sketch by going to: File > Examples > 1.Basics > Blink



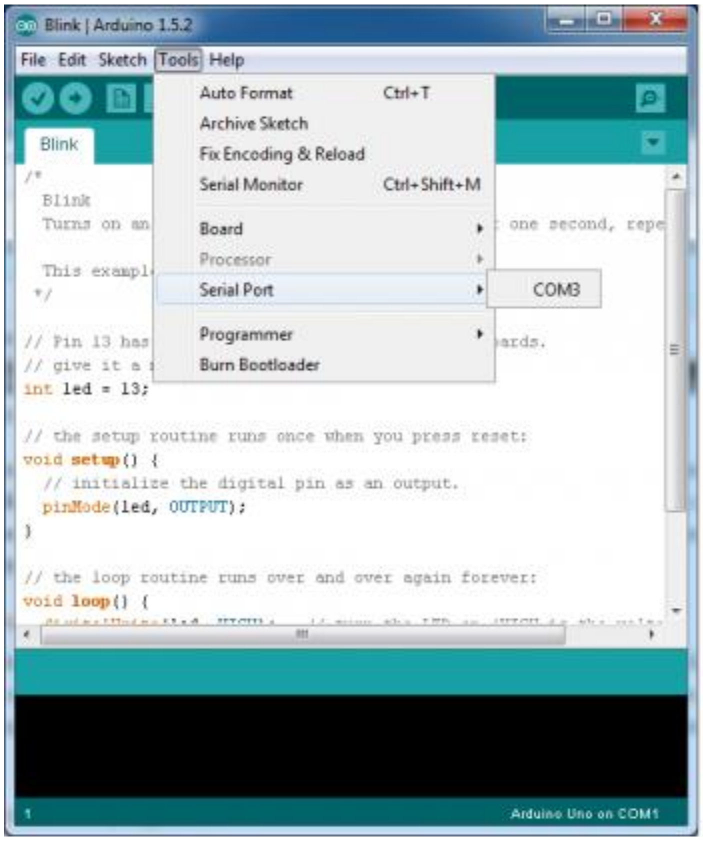
```
}

// the loop routine runs over and ove
void loop() {
```

- Select the type of Arduino board you’re using: Tools > Board > your board type



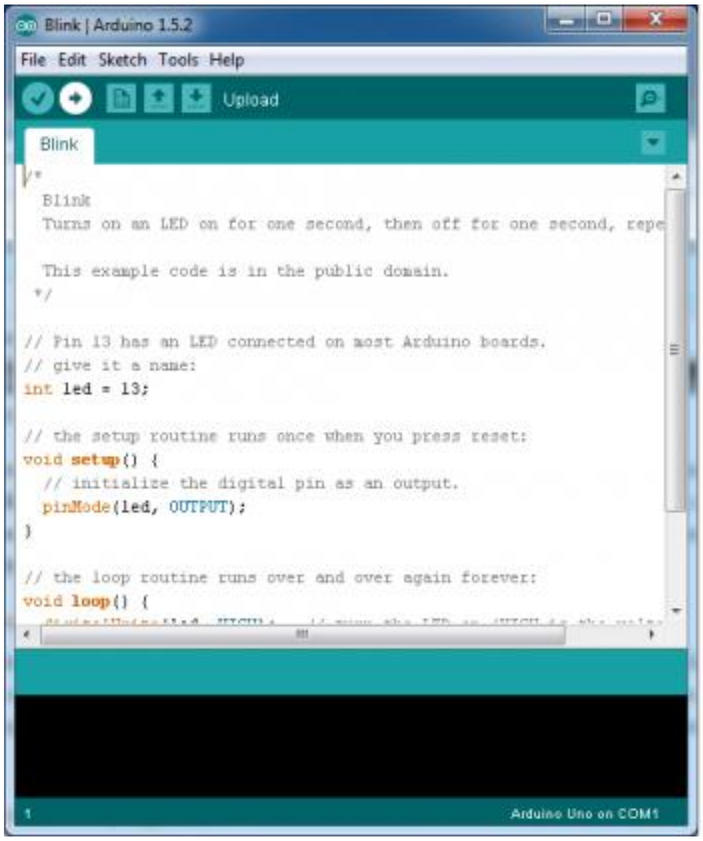
```
/*
  Blink
  Turns on an

  This exampl
 */

// Pin 13 has
// give it a
int led = 13;

// the setup routine runs once when you press r
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again fo
void loop() {
```

- Select the serial/COM port that your Arduino is attached to: Tools > Port > COMxx



```
/*
  Blink
  Turns on an                                  one second, repe

  This exampl
 */

// Pin 13 has                                  ards.
// give it a
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
```

- If you’re not sure which serial device is your Arduino, take a look at the available ports, then unplug your Arduino and look again. The one that disappeared is your Arduino.
- With your Arduino board connected, and the Blink sketch open, press the ‘Upload’ button



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
```

- After a second, you should see some LEDs flashing on your Arduino, followed by the message ‘Done Uploading’ in the status bar of the Blink sketch.
- If everything worked, the onboard LED on your Arduino should now be blinking! You just programmed your first Arduino!

Mac

This page will show you how to install and test the Arduino software on a Mac computer running OSX.

- Go to the Arduino download page and download the latest version of the Arduino software for Mac.
- When the download is finished, un-zip it and open up the Arduino folder to confirm that yes, there are indeed some files and sub-folders inside. The file structure is important so don’t be moving any files around unless you really know what you’re doing.
- Power up your Arduino by connecting your Arduino board to your computer with a USB cable (or FTDI connector if you’re using an Arduino pro). You should see the an LED labed ‘ON’ light up.
- Move the Arduino application into your Applications folder.

FTDI Drivers

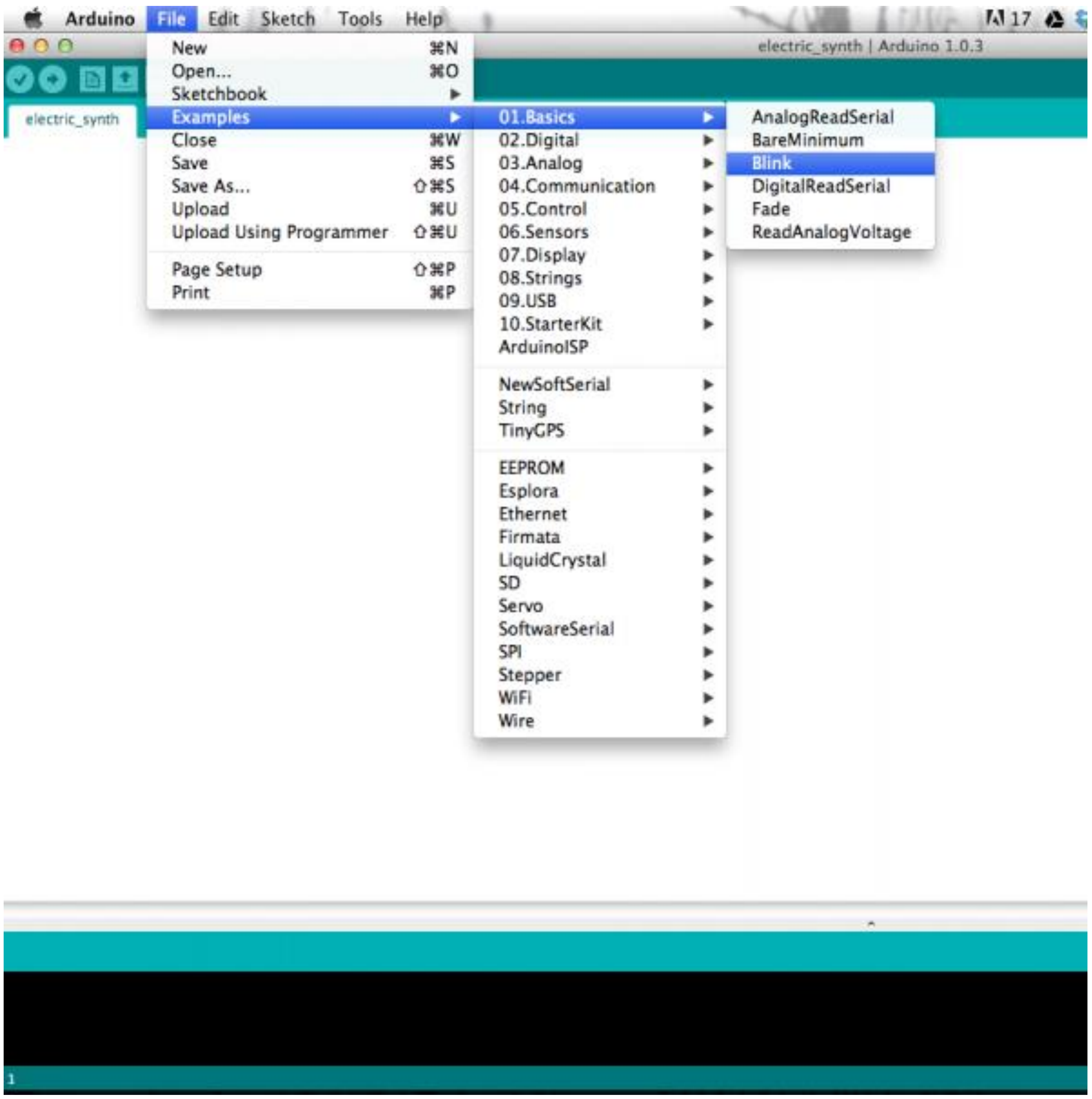
If you have an UNO, Mega2560, or Redboard, you shouldn’t need this step, so skip it!

- For other boards, you will need to install drivers for the FTDI chip on your Arduino.
- Go to the FTDI website and download the latest version of the drivers.
- Once you’re done downloading, double click the package and follow the instructions from the installer.
- Restart your computer after installing the drivers.

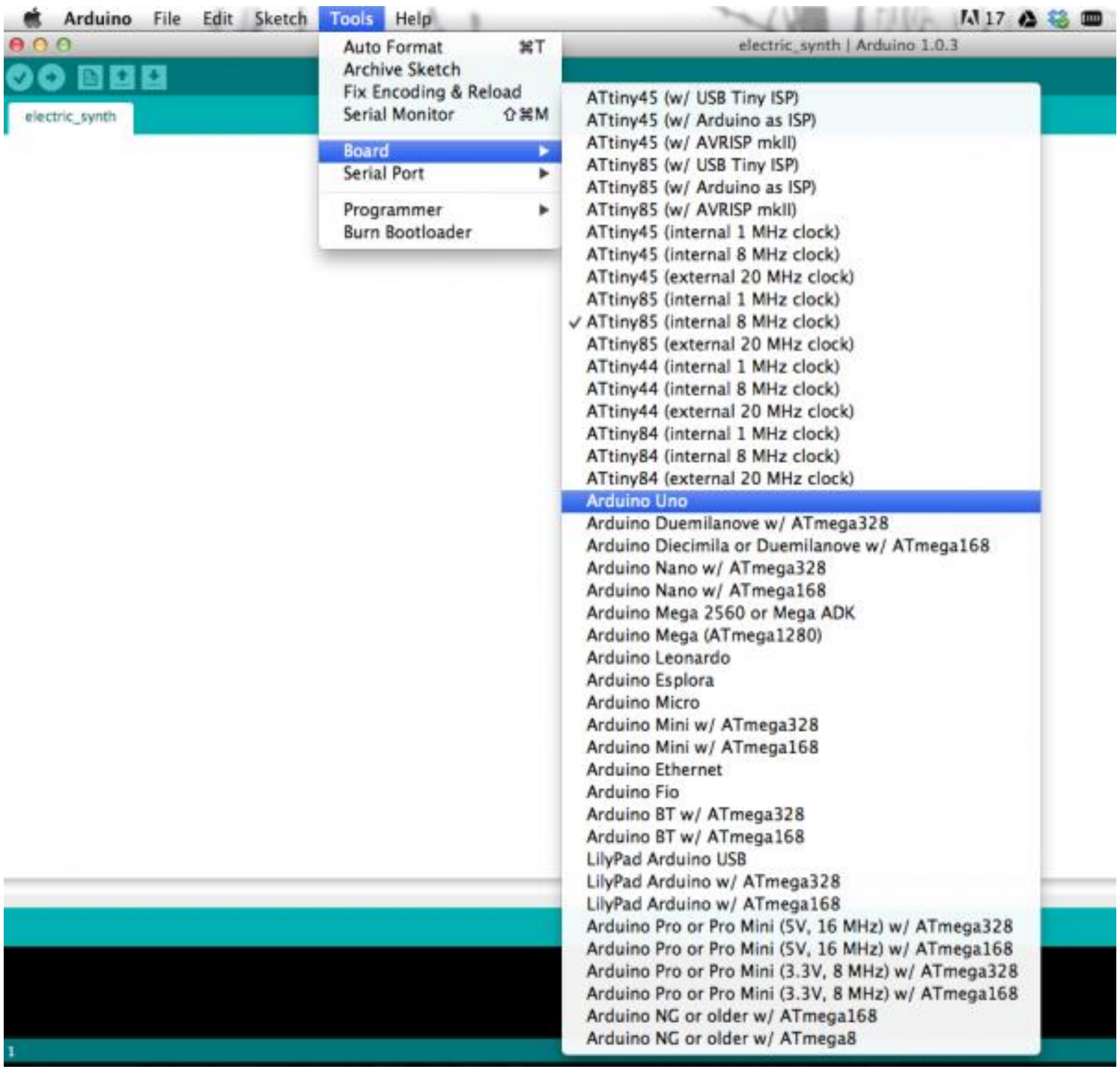
Launch and Blink!

After following the appropriate steps for your software install, we are now ready to test your first program with your Arduino board!

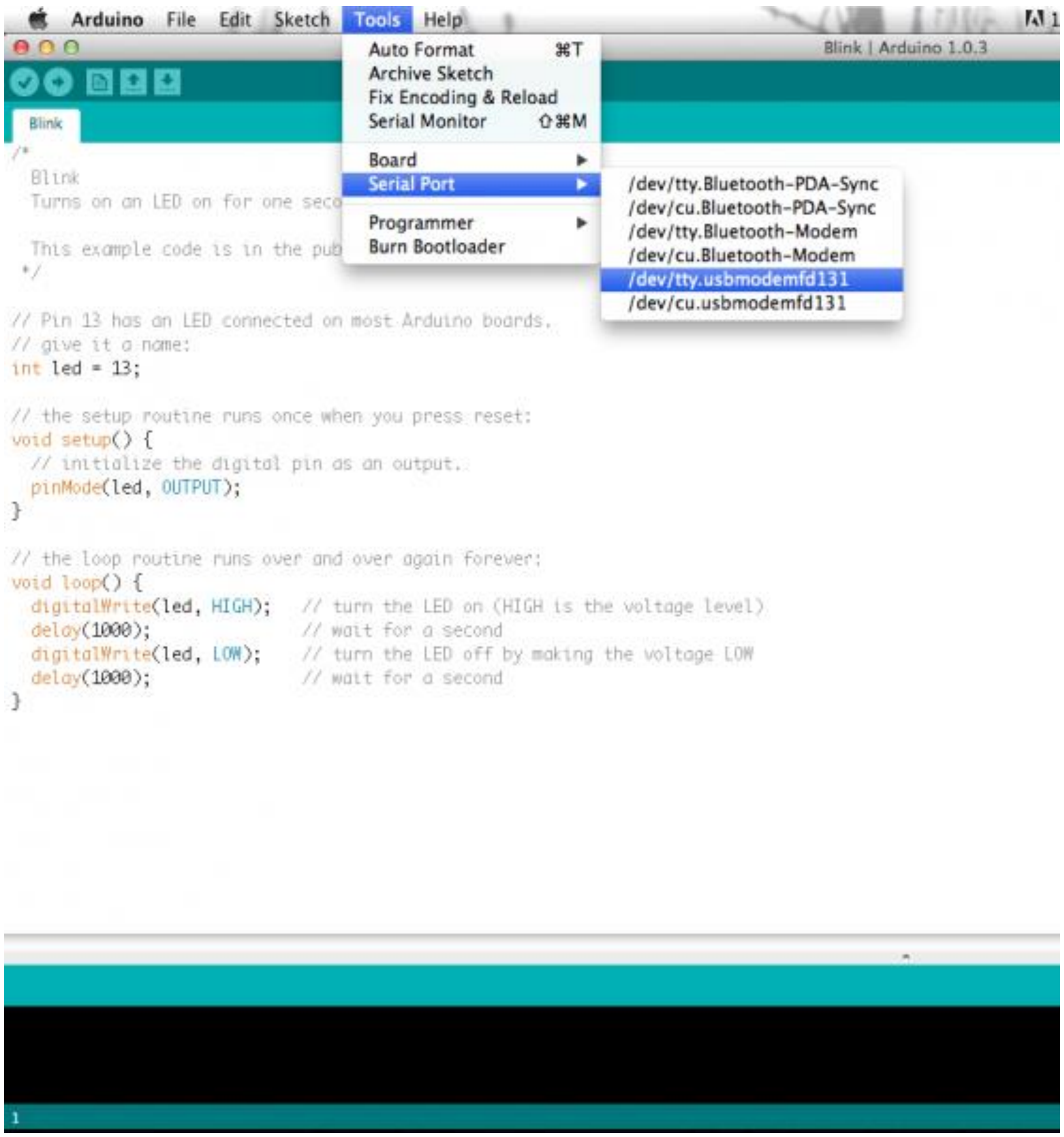
- Launch the Arduino application
- If you disconnected your board, plug it back in
- Open the Blink example sketch by going to: File > Examples > 1.Basics > Blink



- Select the type of Arduino board you’re using: Tools > Board > your board type

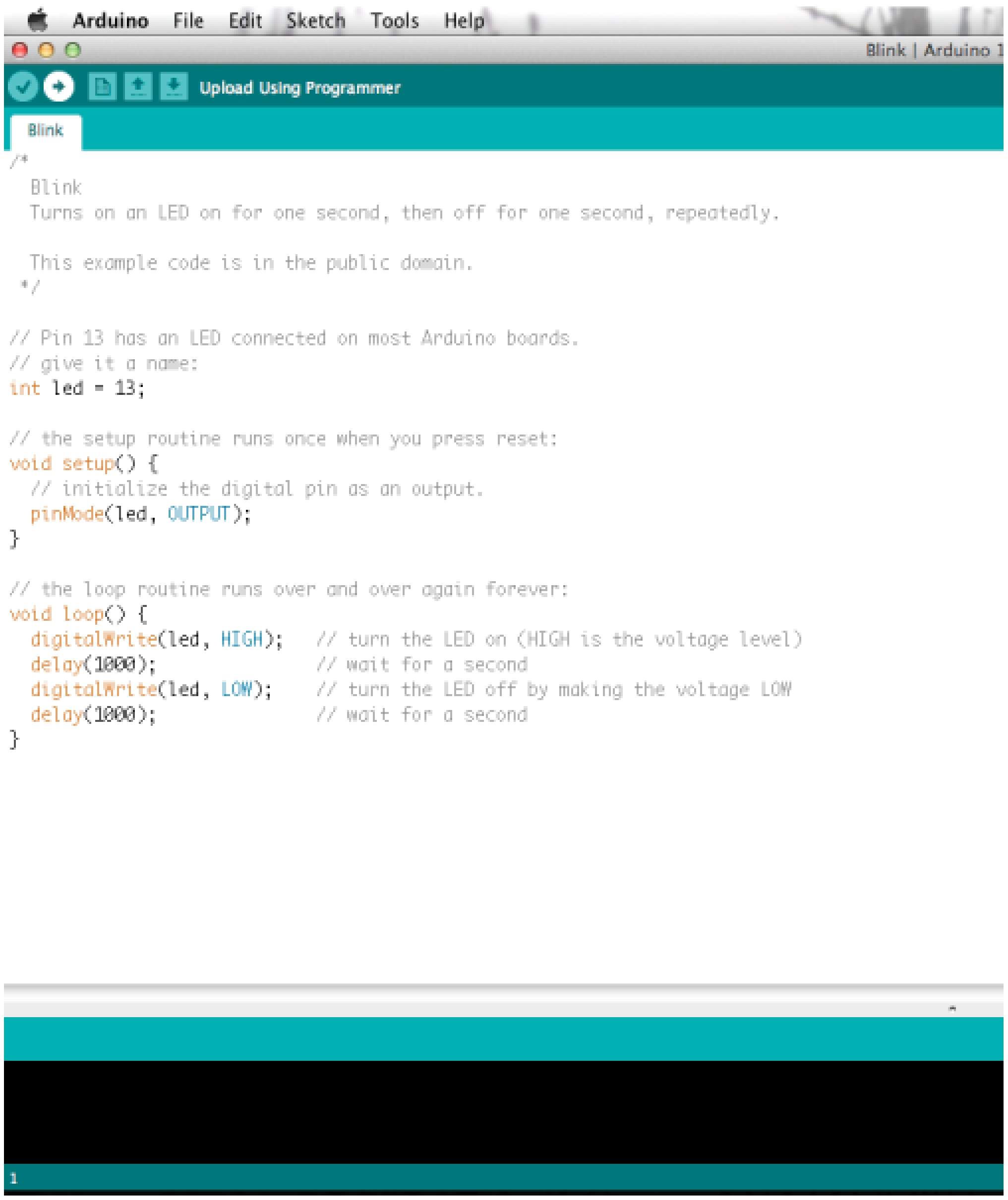

Arduino
File
Edit
Sketch
Tools
Help
electric_synth | Arduino 1.0.3
electric_synth
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Serial Monitor ⇧⌘M
Board
Serial Port
Programmer
Burn Bootloader
ATtiny45 (w/ USB Tiny ISP)
ATtiny45 (w/ Arduino as ISP)
ATtiny45 (w/ AVRISP mkII)
ATtiny85 (w/ USB Tiny ISP)
ATtiny85 (w/ Arduino as ISP)
ATtiny85 (w/ AVRISP mkII)
ATtiny45 (internal 1 MHz clock)
ATtiny45 (internal 8 MHz clock)
ATtiny45 (external 20 MHz clock)
ATtiny85 (internal 1 MHz clock)
✓ ATtiny85 (internal 8 MHz clock)
ATtiny85 (external 20 MHz clock)
ATtiny44 (internal 1 MHz clock)
ATtiny44 (internal 8 MHz clock)
ATtiny44 (external 20 MHz clock)
ATtiny84 (internal 1 MHz clock)
ATtiny84 (internal 8 MHz clock)
ATtiny84 (external 20 MHz clock)
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Esplora
Arduino Micro
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino USB
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
1

- Select the serial port that your Arduino is attached to: Tools > Port > xxxxxx (it'll probably look something like "/dev/tty.usbmodemfd131" or "/dev/tty.usbserial-131" but probably with a different number)



```
/*
  Blink
  Turns on an LED on for one seco

  This example code is in the pub
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

- If you’re not sure which serial device is your Arduino, take a look at the available ports, then unplug your Arduino and look again. The one that disappeared is your Arduino.
- With your Arduino board connected and the Blink sketch open, press the ‘Upload’ button



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

- After a second, you should see some LEDs flashing on your Arduino, followed by the message ‘Done Uploading’ in the status bar of the Blink sketch.
- If everything worked, the onboard LED on your Arduino should now be blinking! You just programmed your first Arduino!

Linux

If you are a Linux user, you probably know that there are many different distribution ‘flavors’ of Linux out there. Unsurprisingly, installing Arduino is slightly different for many of these distributions. Luckily, the Arduino community has done an excellent job of providing instructions for most of the popular versions.

Launch and Blink!

After following the appropriate steps for your software install, we are now ready to test your first program with your Arduino board!

- Launch the Arduino application
- If you disconnected your board, plug it back in
- Open the Blink example sketch by going to: File > Examples > 1.Basics > Blink
- Select the type of Arduino board you're using: Tools > Board > your board type
- Select the serial port that your Arduino is attached to: Tools > Port > xxxxxx (it'll probably look something like "/dev/tty.usbmodemfd131" or "/dev/tty.usbserial-131" but probably with a different number)
- If you're not sure which serial device is your Arduino, take a look at the available ports, then unplug your Arduino and look again. The one that disappeared is your Arduino.
- With your Arduino board connected and the Blink sketch open, press the 'Upload' button
- After a second, you should see some LEDs flashing on your Arduino, followed by the message 'Done Uploading' in the status bar of the Blink sketch.
- If everything worked, the onboard LED on your Arduino should now be blinking! You just programmed your first Arduino!

Troubleshooting

The Arduino Playground Linux section is a great resource for figuring out any problems with your Arduino installation.

Board Add-Ons with Arduino Board Manager

With Arduino v1.6.4+, a new boards manager feature makes it easy to add third-party boards (like the SparkFun Redboard, Digital Sandbox, and RedBot) to the Arduino IDE.

To start, highlight and copy (CTRL + C / CMD + C) the text below for the boards manager URL. You'll need this to configure Arduino.

```
COPY CODEhttps://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_Board_Manager/package_sparkfun_index.json
```

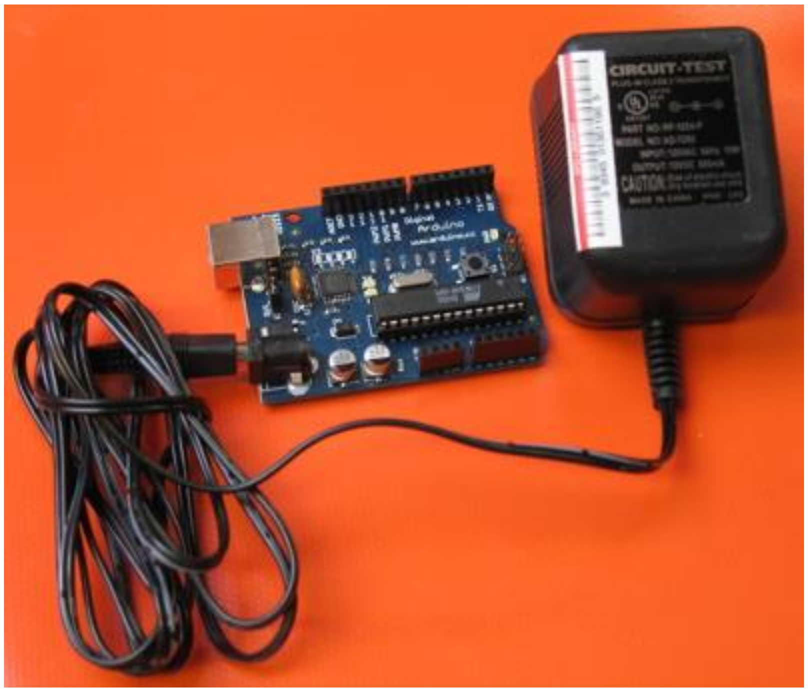
Open up Arduino:

- *Configure the Boards Manager*
  - For Windows and Linux, head to **File>Preferences>Additional Boards Manager URLs** and paste (CTRL + V / CMD + V) the link
  - For Macs, head to **Arduino>Preferences>Additional Boards Manager URLs** and paste (CTRL + V / CMD + V) the link
- Click on **Tools>Board>Boards Manager…**
- Select the Type as "**Contributed**" from the drop down menu.
- Click on the **SparkFun AVR Boards** and then click **Install**

- That’s it! Boards are all installed. This also gives you access to all of our library files as well through the built-in Library Manager Tool in Arduino. Here’s a quick video that goes over the steps in a Windows OS.

Arduino: What Adapter?

Ever since the dawn of Arduino, one question has been asked over and over again: "what kind of DC adapter can I use to power my Arduino?"

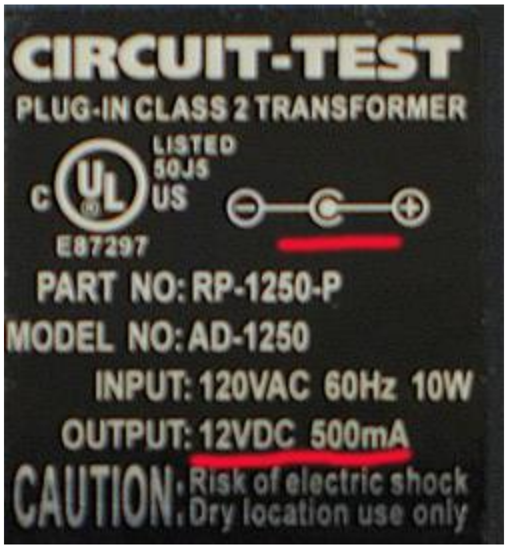


The short answer is: 9 to 12V DC, 250mA or more, 2.1mm plug, centre pin positive.

The long answer is that an off-the shelf Arduino adapter:

- must be a DC adapter (i.e. it has to put out DC, not AC);
- should be between 9V and 12V DC (see note below);
- must be rated for a minimum of 250mA current output, although you will likely want something more like 500mA or 1A output, as it gives you the current necessary to power a servo or twenty LEDs if you want to.
- must have a 2.1mm power plug on the Arduino end, and
- the plug must be "centre positive", that is, the middle pin of the plug has to be the + connection.

These important details are often contained right on the adapter. Here's a picture of an adapter ideally suited to powering the Arduino. The important info is underlined in red here:


CIRCUIT-TEST
PLUG-IN CLASS 2 TRANSFORMER
C UL US LISTED 50JS
E87297
PART NO: RP-1250-P
MODEL NO: AD-1250
INPUT: 120VAC 60Hz 10W
OUTPUT: 12VDC 500mA
CAUTION: Risk of electric shock
Dry location use only

As you can see, it has all the right stuff: 12V, DC, and a little picture that shows you that the middle pin is positive.

Current rating: Since you'll probably be connecting other things to the Arduino (LEDs, LCDs, servos) you should get an adapter that can supply at least 500mA, or even 1000 mA (1 ampere). That way you can be sure you have enough juice to make each component of the circuit function reliably.

One final note. The Arduino's on-board regulator can actually handle up to 20V or more, so you can actually use an adapter that puts out 20V DC. The reasons you don't want to do that are twofold: you'll lose most of that voltage in heat, which is terribly inefficient. Secondly, the nice 9V pin on the Arduino board will actually be putting out 20V or so, which could lead to potential disaster when you connect something expensive to what you thought was the 9V pin. Our advice is to stick with the 9V or 12V DC adapter.

## ARDUINO Programming:

Arduino programs can be divided in three main parts: *structure*, *values* (variables and constants), and *functions*.

```
//
// is a one line Comment
/* this is a
multiline Comment */
```

**Structure**

- setup()
- loop()

These are two specific functions that the Arduino must have. setup() is used to do things you only need done once

**Variables**

**Constants**

- HIGH | LOW

**Data Types**

- char
- int
- float

**pinMode(13,OUTPUT);**
Sets up pin 13 to be an output. Thereby voltage is being sent out by the Arduino through pin 13 to do whatever you want it to do.

Digital Signals
Input of on or off. These signals will always be 5 volts ("HIGH") or 0 Volts or ("LOW")

**digitalWrite()**
Built in function that we use to make an output pin HIGH or LOW. it takes two values; a pin a number, followed by the word HIGH or LOW. Example: digitalWrite(13, HIGH); this sends v Volts through pin 13.

**delay()**
Function that pauses for a given amount of time. Takes one value of time to delay in milliseconds. Example: delay(1000); delays for 1 second.

**Potentionmeter**
Control knob. It changes resistance as it is turned.

**Variable**
Named number. Several types: Int signifies an integer. Integers can range from 32768 to 32767.

**Global Variables**
Variables that are declared outside of a function, and can be seen by all functions.

**analogRead()**

Arduino can read external voltages on the analog input pins. It returns an integer of 0 ( 0 Volts) to 1023 (5 volts)

**RGB**
Red Green Blue

**const**
Putting this in front of a variable means that it will never change once given a value.

**for() Loops**
Three statements separated by semicolons:
1. something to do before starting
2. A test to perform; as long as it's true, it will keep looping
3. Something to do after each loop (Usually increase a variable.)

```
for (x = 0; x < 768; x++)
```

**++**
add one two it. same as x=x+1

**base numbers for color**

0 = pure red
255=pure green
511=pure blue
767 = pure red

**if else statements**

```
if (This statement is true)
{
then do this;
}
else (the if statement is not true)
{
then do this;
}
```

**Serial.println();**
Built in function that you can use in you sketches to send a message back to the serial monitor.

Serial.println(x); ==> x shows up on the computer screen

**Serial.begin(9600);**
Opens the serial port and sets data rate to 9600 bps

**#define**
Allows you to associate a value with a name. Everywhere that this name appears in your sketch, the value will be substituted before the sketch is compiled.
Example: pin assignment: #define ledPin 13
use only if you need to save memory: #define does not use memory

**Empty parentheses ()**
function takes no parameters

**static**
When static is used in front of a variable declaration in a function, it has the effect of initializing the variable only the first time that the function is run.

**return statement**
Any non-void function has to have a return statement in it. int functions return ints

**float variables**
Variables with a decimal point

**boolean**
Type of logic, more later.

**&&**
Boolean operator "and"

**||**
Boolean operator "or"

**!**
Not