

6-Intro to Arduino

Intro to Arduino

Introduction: Intro to Arduino



An Arduino is an open-source microcontroller development board. In plain English, you can use the Arduino to read sensors and control things like motors and lights. This allows you to upload programs to this board which can then interact with things in the real world. With this, you can make devices which respond and react to the world at large.

For instance, you can read a humidity sensor connected to a potted plant and turn on an automatic watering system if it gets too dry. Or, you can make a stand-alone chat server which is plugged into your internet router. Or, you can have it tweet every time your cat passes through a pet door. Or, you can have it start a pot of coffee when your alarm goes off in the morning.

Basically, if there is something that is in any way controlled by

electricity, the Arduino can interface with it in some manner. And even if it is not controlled by electricity, you can probably still use things which are (like motors and electromagnets), to interface with it.

The possibilities of the Arduino are almost limitless. As such, there is no way that one single tutorial can cover everything you might ever need to know. That said, I've done my best to give a basic overview of the fundamental skills and knowledge that you need to get your Arduino up and running. If nothing more, this should function as a springboard into further experimentation and learning.

Step 1: Different Types of Arduinos





There are a number of different types of Arduinos to choose from. This is a brief overview of some of the more common types of Arduino boards you may encounter. For a full listing of currently support Arduino boards, check out the [Arduino hardware page](#).

Arduino Uno

The most common version of Arduino is the [Arduino Uno](#). This board is what most people are talking about when they refer to

an Arduino. In the next step, there is a more complete rundown of its features.

Arduino NG, Diecimila, and the Duemilanove (Legacy Versions)

Legacy versions of the Arduino Uno product line consist of the NG, Diecimila, and the Duemilanove. The important thing to note about legacy boards is that they lack particular feature of the Arduino Uno. Some key differences:

- The Diecimila and NG use an ATMEGA168 chips (as opposed to the more powerful ATMEGA328),
- Both the Diecimila and NG have a jumper next to the USB port and require manual selection of either USB or battery power.
- The Arduino NG requires that you hold the reset button on the board for a few seconds prior to uploading a program.

Arduino Mega 2560

The [Arduino Mega 2560](#) is the second most commonly encountered version of the Arduino family. The Arduino Mega is like the Arduino Uno's beefier older brother. It boasts 256 KB of memory (8 times more than the Uno). It also has 54 input and output pins, 16 of which are analog pins, and 14 of which can do PWM. However, all of the added functionality comes at the cost of a slightly larger circuit board. It may make your project more powerful, but it will also make your project larger. Check out the official [Arduino Mega 2560 page](#) for more details.

Arduino Mega ADK

This specialized version of the Arduino is basically an Arduino Mega that has been specifically designed for interfacing with Android smartphones. This too is now a legacy version.

Arduino Yun

The [Arduino Yun](#) uses a ATmega32U4 chip instead of the ATmega328. However, what really sets it apart is the addition of the Atheros AR9331 microprocessor. This extra chip allows this board to run Linux in addition to the normal Arduino operating system. If all of that were not enough, it also has onboard wifi capability. In other words, you can program the board to do stuff like you would with any other Arduino, but you can also access the Linux side of the board to connect to the internet via wifi. The Arduino-side and Linux-side can then easily communicate back and forth with each other. This makes this board extremely powerful and versatile. I'm barely scratching the surface of what you can do with this, but to learn more, check out the official [Arduino Yun page](#).

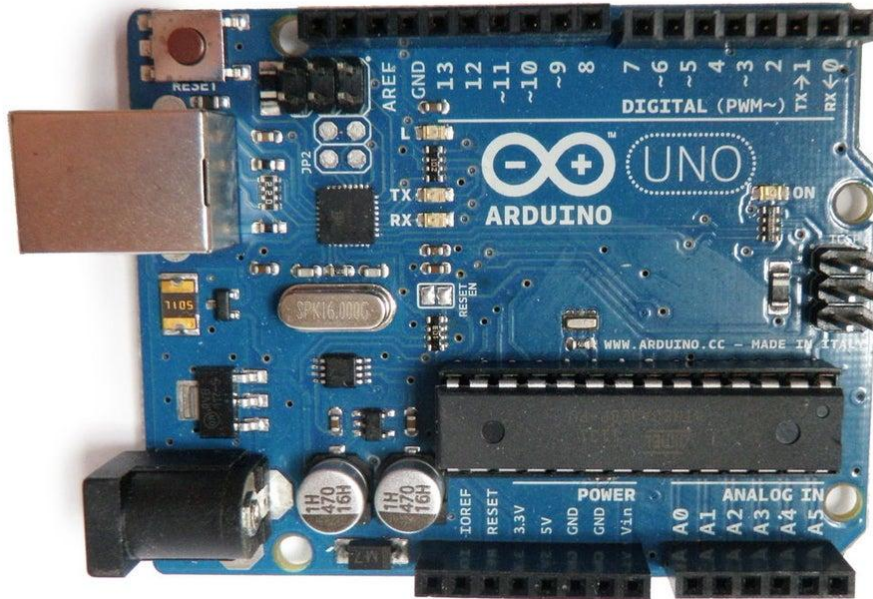
Arduino Nano

If you want to go smaller than the standard Arduino board, the [Arduino Nano](#) is for you! Based on a surface mount ATmega328 chip, this version of the Arduino has been shrunk down to a small footprint capable of fitting into tight spaces. It can also be inserted directly into a breadboard, making it easy to prototype with.

Arduino LilyPad

The [LilyPad](#) was designed for wearable and e-textile applications. It is intended to be sewn to fabric and connected to other sewable components using conductive thread. This board requires the use of a special [FTDI-USB TTL serial programming cable](#). For more information, the [Arduino LilyPad page](#) is a decent starting point.

Step 2: Arduino Uno Features



Some people think of the entire Arduino board as a microcontroller, but this is inaccurate. The Arduino board actually is a specially designed circuit board for programming and prototyping with Atmel microcontrollers.

The nice thing about the Arduino board is that it is relatively cheap, plugs straight into a computer's USB port, and it is dead-simple to setup and use (compared to other development boards).

Some of the key features of the Arduino Uno include:

- An open source design. The advantage of it being open source is that it has a large community of people using and troubleshooting it. This makes it easy to find someone to help you debug your projects.

- An easy USB interface . The chip on the board plugs straight into your USB port and registers on your computer as a virtual serial port. This allows you to interface with it as through it were a serial device. The benefit of this setup is that serial communication is an extremely easy (and time-tested) protocol, and USB makes connecting it to modern computers really convenient.
- Very convenient power management and built-in voltage regulation. You can connect an external power source of up to 12v and it will regulate it to both 5v and 3.3v. It also can be powered directly off of a USB port without any external power.
- An easy-to-find, and dirt cheap, microcontroller "brain." The ATmega328 chip retails for about \$2.88 on Digikey. It has countless number of nice hardware features like timers, PWM pins, external and internal interrupts, and multiple sleep modes. Check out the official [datasheet](#) for more details.
- A 16mhz clock. This makes it not the speediest microcontroller around, but fast enough for most applications.
- 32 KB of flash memory for storing your code.
- 13 digital pins and 6 analog pins. These pins allow you to connect external hardware to your Arduino. These pins are key for extending the computing capability of the Arduino into the real

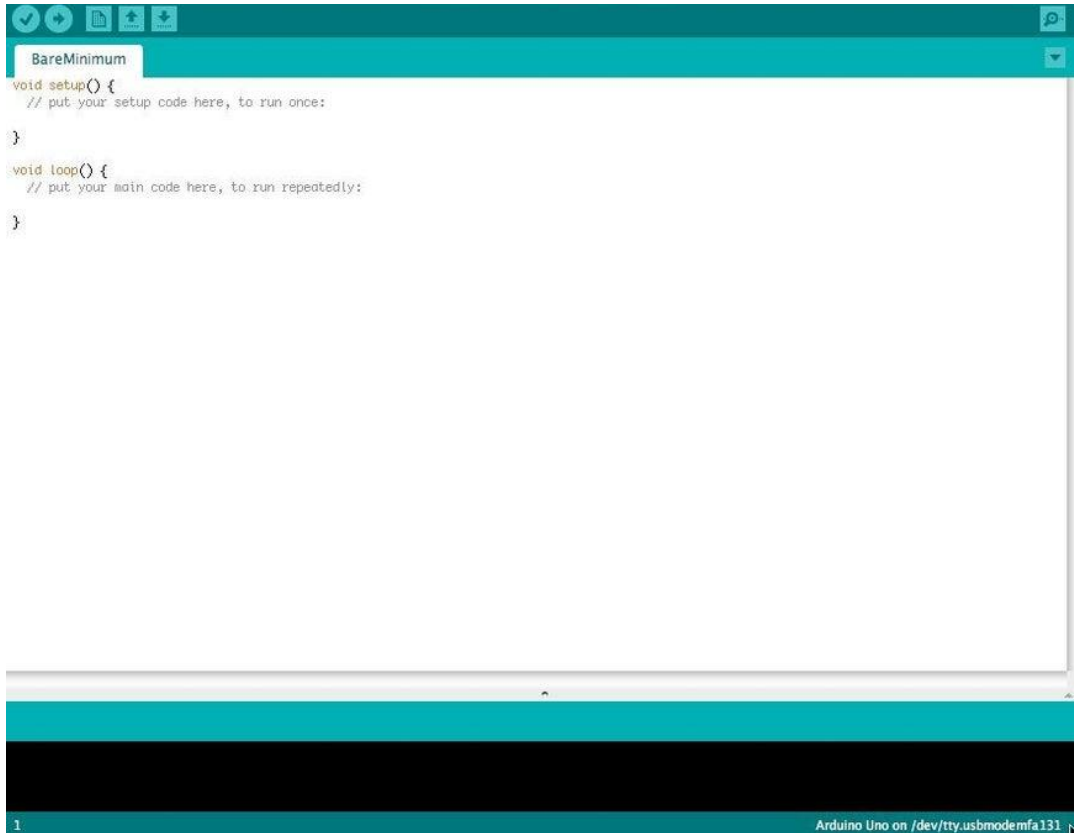
world. Simply plug your devices and sensors into the sockets that correspond to each of these pins and you are good to go.

- An ICSP connector for bypassing the USB port and interfacing the Arduino directly as a serial device. This port is necessary to [re-bootload your chip](#) if it corrupts and can no longer talk to your computer.
- An on-board LED attached to digital pin 13 for fast and easy debugging of code.
- And last, but not least, a button to reset the program on the chip.

For a complete rundown of all the Arduino Uno has to offer, be sure to check out the [official Arduino page](#).

Add TipAsk QuestionComment

Step 3: Arduino IDE



Before you can start doing anything with the Arduino, you need to download and install the [Arduino IDE](#) (integrated development environment). From this point on we will be referring to the Arduino IDE as the Arduino Programmer.

The Arduino Programmer is based on the [Processing IDE](#) and uses a variation of the C and C++ programming languages.

Step 4: Plug It In



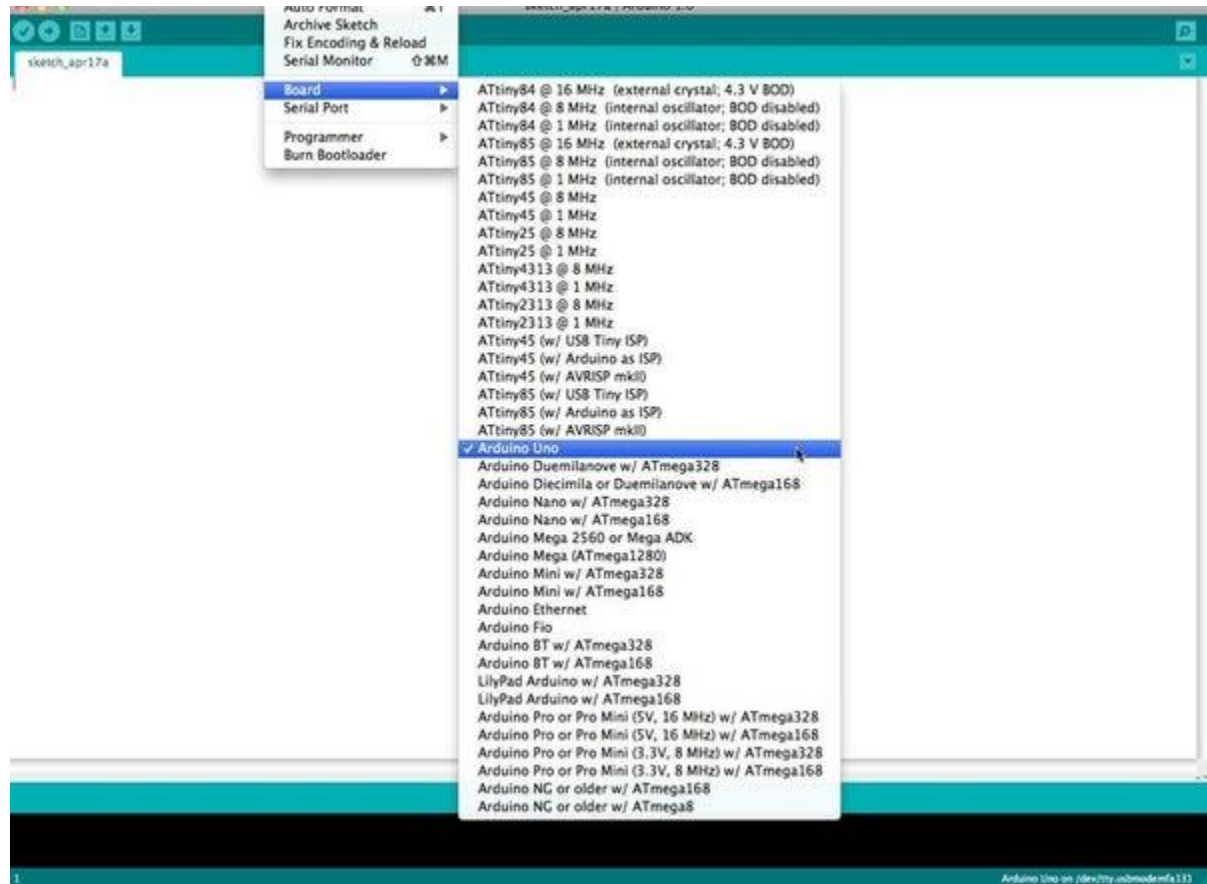
Connect the Arduino to your computer's USB port.

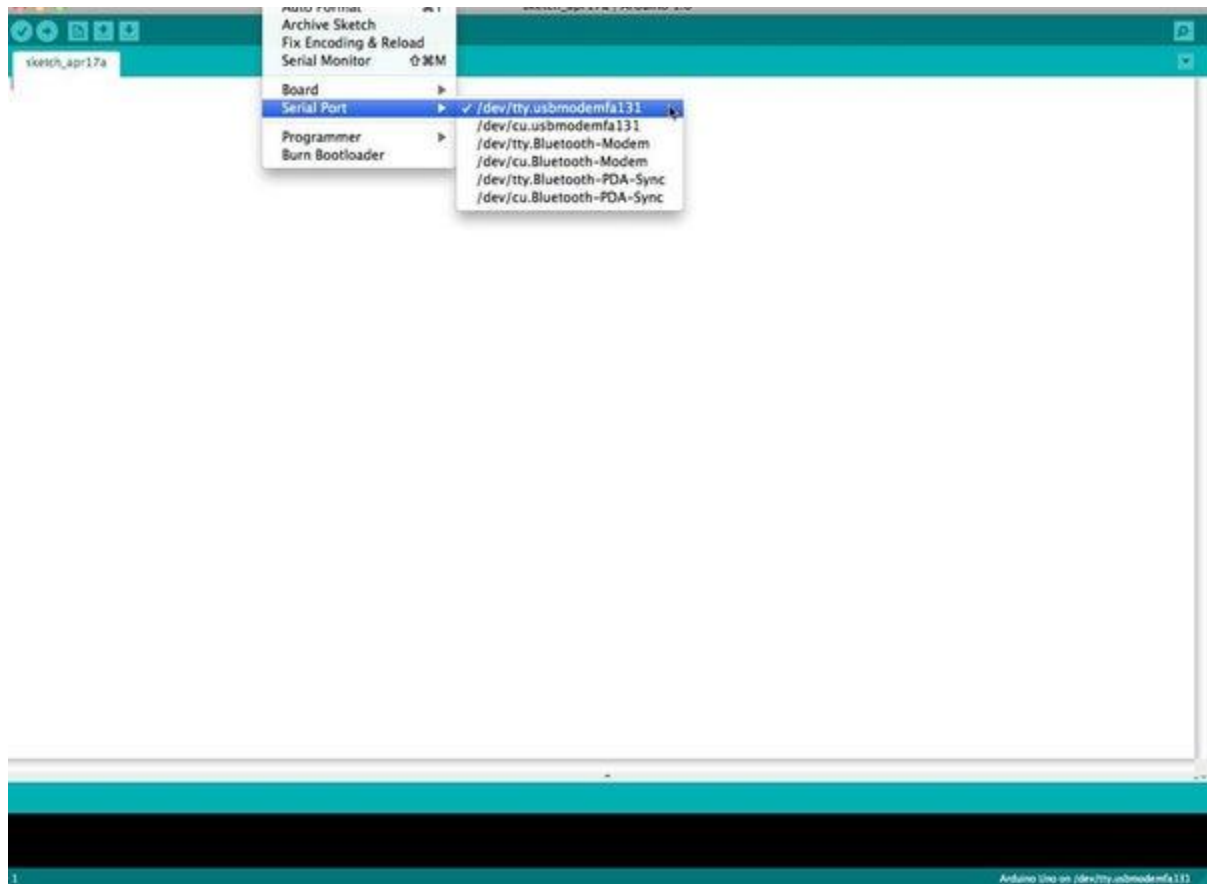
Please note that although the Arduino plugs into your computer, it is not a true USB device. The board has a special chip that allows it to show up on your computer as a virtual serial port when it is plugged into a USB port. This is why it is important to plug the board in. When the board is not plugged in, the virtual serial port that the Arduino operates upon will not be present (since all of the information about it lives on the Arduino board).

It is also good to know that every single Arduino has a unique virtual serial port address. This means that every time you plug in a different Arduino board into your computer, you will need to reconfigure the serial port that is in use.

The Arduino Uno requires a [male USB A to male USB B cable](#).

Step 5: Settings





Before you can start doing anything in the Arduino programmer, you must set the board-type and serial port.

To set the board, go to the following:

Tools --> Boards

Select the version of board that you are using. Since I have an Arduino Uno plugged in, I obviously selected "Arduino Uno."

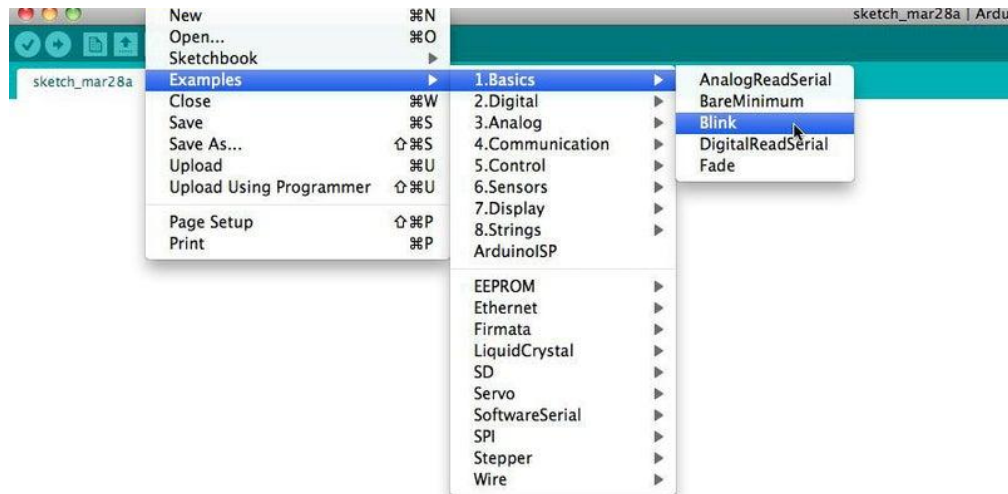
To set the serial port, go to the following:

Tools --> Serial Port

Select the serial port that looks like:

/dev/tty.usbmodem [random numbers]

Step 6: Run a Sketch





Arduino programs are called sketches. The Arduino programmer comes with a ton of example sketches preloaded. This is great because even if you have never programmed anything in your life, you can load one of these sketches and get the Arduino to do something.

To get the LED tied to digital pin 13 to blink on and off, let's load the blink example.

The blink example can be found here:

Files --> Examples --> Basics --> Blink

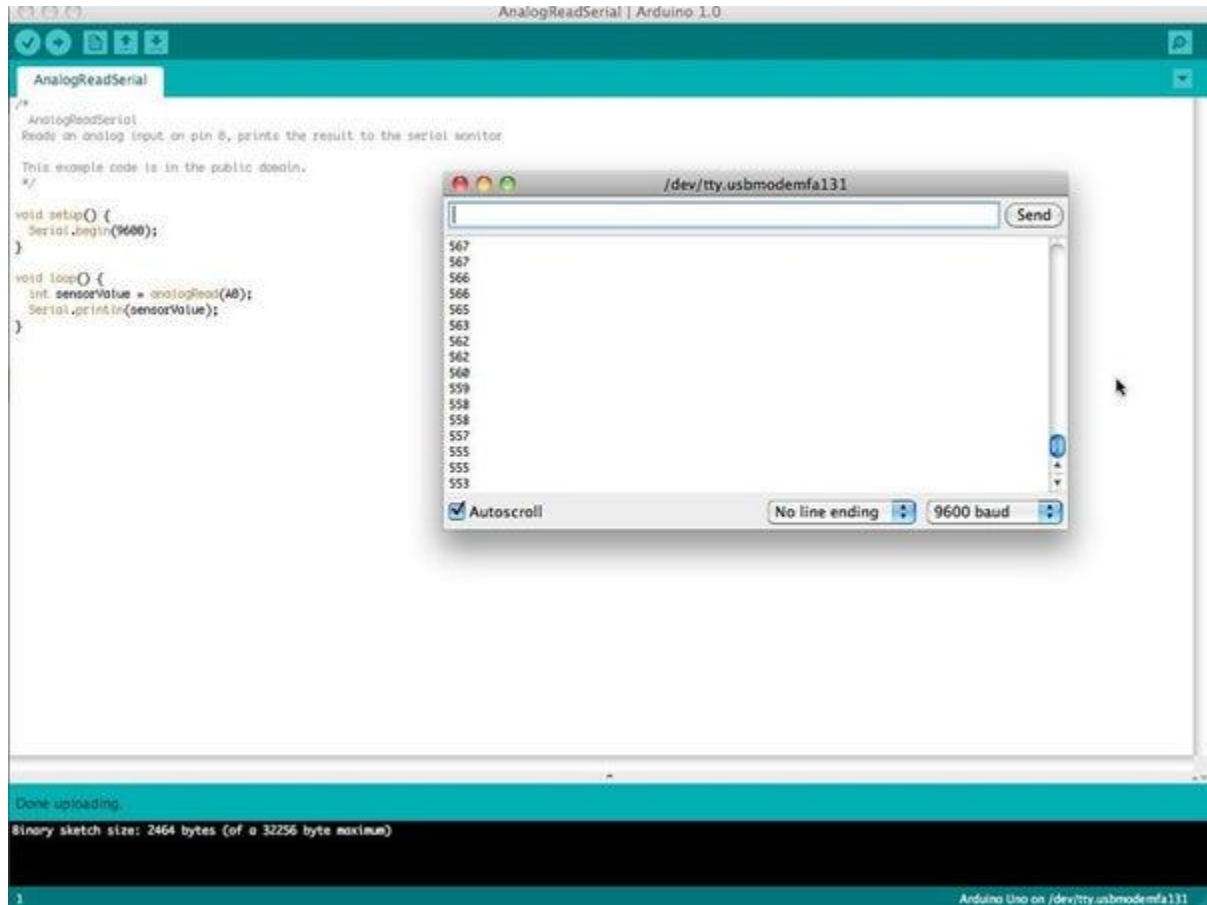
The blink example basically sets pin D13 as an output and then blinks the test LED on the Arduino board on and off every second.

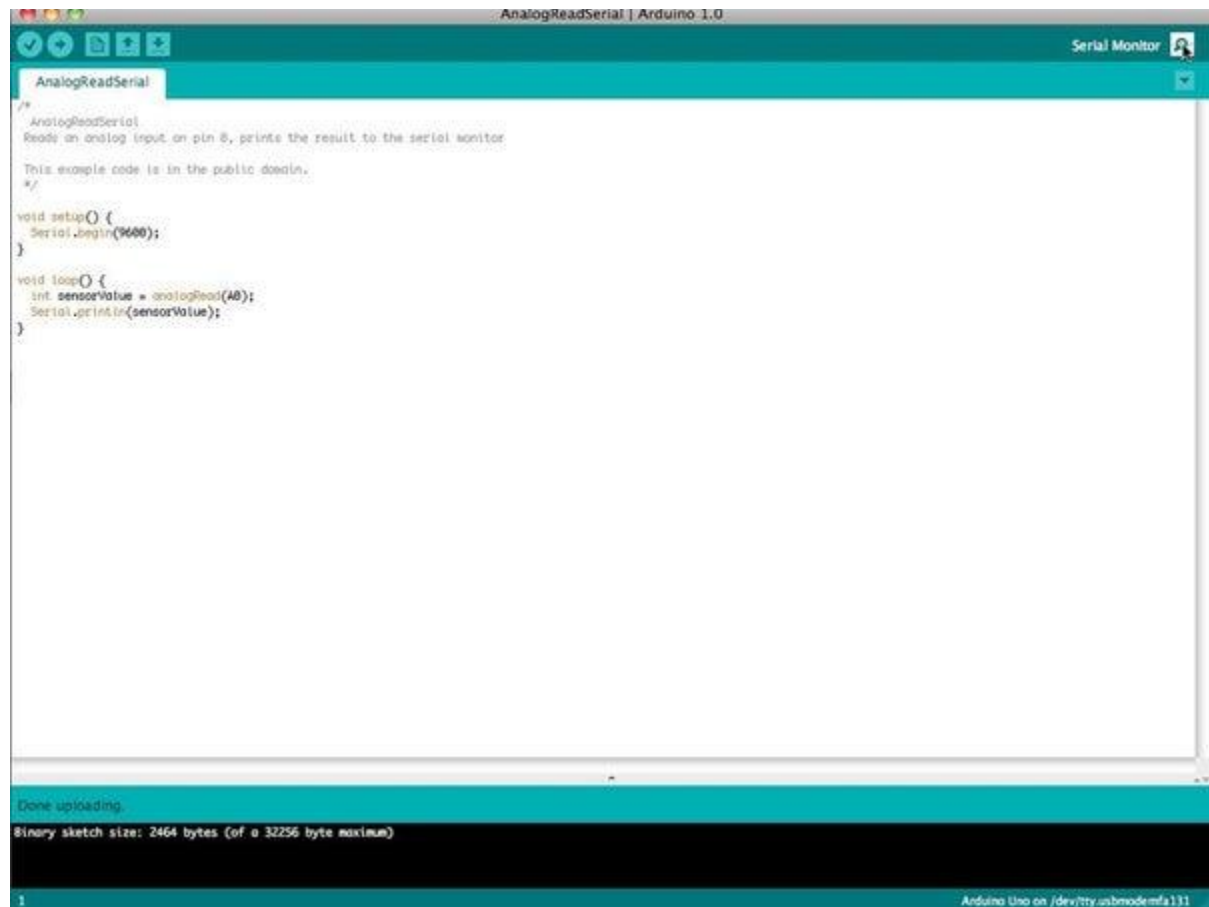
Once the blink example is open, it can be installed onto the ATMEGA328 chip by pressing the upload button, which looks like an arrow pointing to the right.

Notice that the surface mount status LED connected to pin 13 on the Arduino will start to blink. You can change the rate of the blinking by changing the length of the delay and pressing the upload button again.

Add TipAsk QuestionComment

Step 7: Serial Monitor





The serial monitor allows your computer to connect serially with the Arduino. This is important because it takes data that your Arduino is receiving from sensors and other devices and displays it in real-time on your computer. Having this ability is invaluable to debug your code and understand what number values the chip is actually receiving.

For instance, connect center sweep (middle pin) of a potentiometer to A0, and the outer pins, respectively, to 5v and ground. Next upload the sketch shown below:

File --> Examples --> 1.Basics --> AnalogReadSerial

Click the button to engage the serial monitor which looks like a magnifying glass. You can now see the numbers being read by

the analog pin in the serial monitor. When you turn the knob the numbers will increase and decrease.

The numbers will be between the range of 0 and 1023. The reason for this is that the analog pin is converting a voltage between 0 and 5V to a discrete number.

Step 8: Digital In



The Arduino has two different types of input pins, those being analog and digital.

To begin with, let's look at the digital input pins.

Digital input pins only have two possible states, which are on or off. These two on and off states are also referred to as:

- HIGH or LOW

- 1 or 0
- 5V or 0V.

This input is commonly used to sense the presence of voltage when a switch is opened or closed.

Digital inputs can also be used as the basis for countless digital communication protocols. By creating a 5V (HIGH) pulse or 0V (LOW) pulse, you can create a binary signal, the basis of all computing. This is useful for talking to digital sensors like a PING ultrasonic sensor, or communicating with other devices.

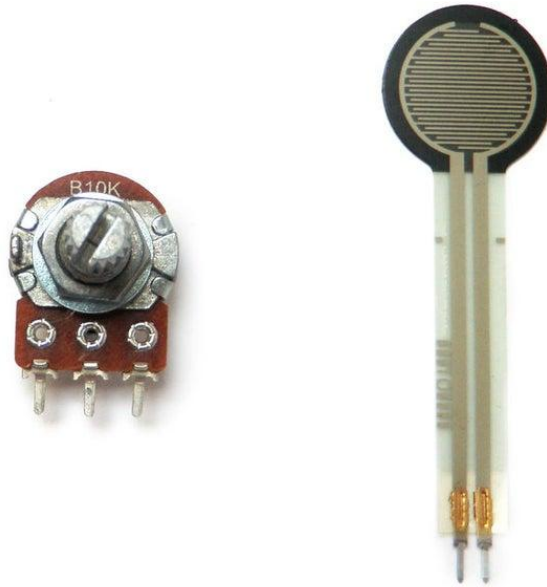
For a simple example of a digital input in use, connect a switch from digital pin 2 to 5V, a 10K resistor** from digital pin 2 to ground, and run the following code:

File --> Examples --> 2.Digital --> Button

**The 10K resistor is called a pull-down resistor because it connects the digital pin to ground when the switch is not pressed. When the switch is pressed, the electrical connections in the switch has less resistance than the resistor, and the electricity no longer connects to ground. Instead, electricity flows between 5V and the digital pin. This is because electricity always chooses the path of least resistance. To learn more about this, visit the [Digital Pins](#) page.

Add TipAsk QuestionComment

Step 9: Analog In



Aside from the digital input pins, the Arduino also boasts a number of analog input pins.

Analog input pins take an analog signal and perform a 10-bit analog-to-digital (ADC) conversion to turn it into a number between 0 and 1023 (4.9mV steps).

This type of input is good for reading resistive sensors. These are basically sensors which provide resistance to the circuit. They are also good for reading a varying voltage signal between 0 and 5V. This is useful when interfacing with various types of analog circuitry.

If you followed the example in Step 7 for engaging the serial monitor, you have already tried using an analog input pin.

[Add Tip](#)[Ask Question](#)[Comment](#)

Step 10: Digital Out



A digital out pin can be set to be HIGH (5v) or LOW (0v). This allows you to turn things on and off.

Aside from turning things on and off (and making LEDs blink), this form of output is convenient for a number of applications.

Most notably, it allows you to communicate digitally. By turning the pin on and off rapidly, you are creating binary states (0 and 1), which is recognized by countless other electronic devices as a binary signal. By using this method, you can communicate using a number of different protocols.

Digital communication is an advanced topic, but to get a general idea of what can be done, check out the [Interfacing With Hardware](#) page.

If you followed the example in Step 6 for getting an LED to blink, you have already tried using a digital output pin.

Add TipAsk QuestionComment

Step 11: Analog Out



As mentioned earlier, the Arduino has a number of built in special functions. One of these special functions is pulse width modulation, which is the way an Arduino is able to create an analog-like output.

Pulse width modulation - or PWM for short - works by rapidly turning the PWM pin high (5V) and low (0V) to simulate an analog signal. For instance, if you were to blink an LED on and off rapidly enough (about five milliseconds each), it would seem to average the brightness and only appear to be receiving half the power. Alternately, if it were to blink on for 1 millisecond and then blink off for 9 millisecond, the LED would appear to be 1/10 as bright and only be receiving 1/10 the voltage.

PWM is key for a number of applications including making sound, controlling the brightness of lights, and controlling the speed of motors.

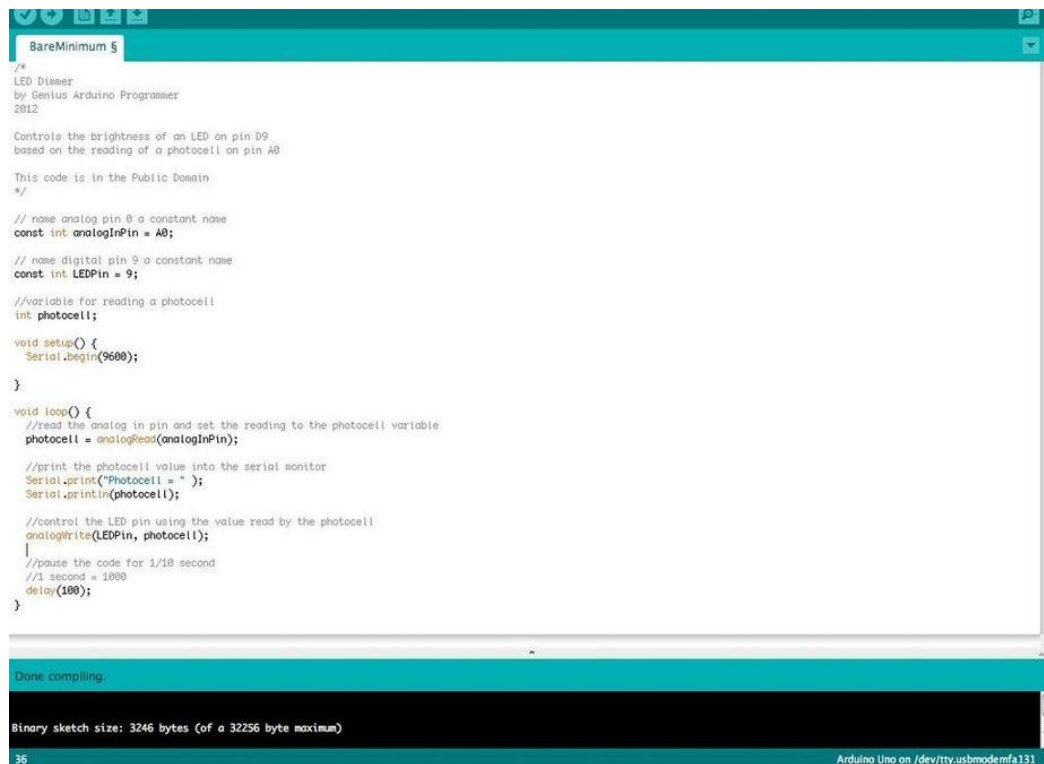
.

To try out PWM yourself, connect an LED and 220 ohm resistor to digital pin 9, in series to ground. Run the following example code:

File --> Examples --> 3.Analog --> Fading

Add TipAsk QuestionComment

Step 12: Write Your Own Code



```
/*
  BareMinimum
  LED Dimmer
  by Genius Arduino Programmer
  2012

  Controls the brightness of an LED on pin D9
  based on the reading of a photocell on pin A0

  This code is in the Public Domain
  */

// name analog pin 0: a constant name
const int analogInPin = A0;

// name digital pin 9: a constant name
const int LEDPin = 9;

//variable for reading a photocell
int photocell;

void setup() {
  Serial.begin(9600);
}

void loop() {
  //read the analog in pin and set the reading to the photocell variable
  photocell = analogRead(analogInPin);

  //print the photocell value into the serial monitor
  Serial.print("Photocell = ");
  Serial.println(photocell);

  //control the LED pin using the value read by the photocell
  analogWrite(LEDPin, photocell);
  //pause the code for 1/10 second
  //1 second = 1000
  delay(100);
}
```

Done compiling.

Binary sketch size: 3246 bytes (of a 32256 byte maximum)

36 Arduino Uno on /dev/tty.usbmodemfa131

To write your own code, you will need to learn some basic programming language syntax. In other words, you have to learn how to properly form the code for the programmer to understand it. You can think of this kind of like understanding grammar and punctuation. You can write an entire book without proper grammar and punctuation, but no one will be able to understand it, even if it is in English.

Some important things to keep in mind when writing your own code:

- An Arduino program is called a sketch.
- All code in an Arduino sketch is processed from top to bottom.
- Arduino sketches are typically broken into five parts.

1. The sketch usually starts with a header that explains what the sketch is doing, and who wrote it.
2. Next, it usually defines global variables. Often, this is where constant names are given to the different Arduino pins.
3. After the initial variables are set, the Arduino begins the setup routine. In the setup function, we set initial conditions of variables when necessary, and run any preliminary code that we only want to run once. This is where serial communication is initiated, which is required for running the serial monitor.
4. From the setup function, we go to the loop routine. This is the main routine of the sketch. This is not only where your main code goes, but it will be executed over and over, so long as the sketch continues to run.
5. Below the loop routine, there is often other functions listed. These functions are user-defined and only activated when called in the setup and loop routine. When these functions are called, the Arduino processes all of the code in the function from top to bottom and then goes back to the next line in the sketch where it left off when the function was called. Functions are good because they allow you to run standard routines - over and over - without having to write the same lines of code over and over. You can simply call upon a function multiple times, and this will free up memory

on the chip because the function routine is only written once. It also makes code easier to read. To learn how to form your own functions, check out [this page](#).

- All of that said, the only two parts of the sketch which are mandatory are the Setup and Loop routines.
- Code must be written in the [Arduino Language](#), which is roughly based on C.
- Almost all statements written in the Arduino language must end with a ;
- Conditionals (such as [if statements](#) and [for loops](#)) do not need a ;
- Conditionals have their own rules and can be found under "[Control Structures](#)" on the [Arduino Language](#) page
- Variables are storage compartments for numbers. You can pass values into and out of variables. Variables must be defined (stated in the code) before they can be used and need to have a data type associated with it. To learn some of the basic data types, review the [Language Page](#).

Okay! So let us say we want to write code that reads a photocell connected to pin A0, and use the reading we get from the

photocell to control the brightness of an LED connected to pin D9.

First, we want to open the BareMinimum sketch, which can be found at:

File --> Examples --> 1.Basic --> BareMinimum

The BareMinimum Sketch should look like this:

```
<pre>void setup() {  
    // put your setup code here, to run once:  
  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
  
}
```

Next, lets put a header on the code, so other people know about what we are making, why, and under what terms:

```
<pre>/*  
  
LED Dimmer  
  
by Genius Arduino Programmer  
  
2012  
  
  
Controls the brightness of an LED on pin D9  
based on the reading of a photocell on pin A0
```

This code is in the Public Domain

*/

void setup() {

 // put your setup code here, to run once:

}

void loop() {

 // put your main code here, to run repeatedly:

}

Once that is all squared away, let us define the pin names, and establish variables:

<pre>/*

LED Dimmer

by Genius Arduino Programmer

2012

Controls the brightness of an LED on pin D9

based on the reading of a photocell on pin A0

This code is in the Public Domain

*/


```
// name analog pin 0 a constant name
const int analogInPin = A0;

// name digital pin 9 a constant name
const int LEDPin = 9;

//variable for reading a photocell
int photocell;

void setup() {
    // put your setup code here, to run once:

}

void loop() {
    // put your main code here, to run repeatedly:

}
```

Now that variables and pin names are set, let us write the actual code:

```
<pre>/*
LED Dimmer
by Genius Arduino Programmer
2012

Controls the brightness of an LED on pin D9
```

based on the reading of a photocell on pin A0

This code is in the Public Domain

```
*/
```

```
// name analog pin 0 a constant name
```

```
const int analogInPin = A0;
```

```
// name digital pin 9 a constant name
```

```
const int LEDPin = 9;
```

```
//variable for reading a photocell
```

```
int photocell;
```

```
void setup() {
```

```
//nothing here right now
```

```
}
```

```
void loop() {
```

```
    //read the analog in pin and set the reading to the photocell variable
```

```
    photocell = analogRead(analogInPin);
```

```
    //control the LED pin using the value read by the photocell
```

```
    analogWrite(LEDPin, photocell);
```

```
    //pause the code for 1/10 second
```

```
    //1 second = 1000
```

```
    delay(100);  
}
```

If we want to see what numbers the analog pin is actually reading from the photocell, we will need to use the serial monitor. Let's activate the serial port and output those numbers:

```
<pre>/*  
LED Dimmer  
by Genius Arduino Programmer  
2012  
  
Controls the brightness of an LED on pin D9  
based on the reading of a photocell on pin A0  
  
This code is in the Public Domain  
*/  
  
// name analog pin 0 a constant name  
const int analogInPin = A0;  
  
// name digital pin 9 a constant name  
const int LEDPin = 9;  
  
//variable for reading a photocell  
int photocell;  
  
void setup() {
```

```
Serial.begin(9600);

}

void loop() {
  //read the analog in pin and set the reading to the photocell variable
  photocell = analogRead(analogInPin);

  //print the photocell value into the serial monitor
  Serial.print("Photocell = " );
  Serial.println(photocell);

  //control the LED pin using the value read by the photocell
  analogWrite(LEDpin, photocell);

  //pause the code for 1/10 second
  //1 second = 1000
  delay(100);
}
```

For more information about formulating code, visit the [Foundations Page](#). If you need help with the Arduino Language, then the [Language Page](#) is the place for you.

Also, the [Example Sketch Page](#) is a great place to start messing around with code. Don't be afraid to change things and experiment.

Step 13: Shields



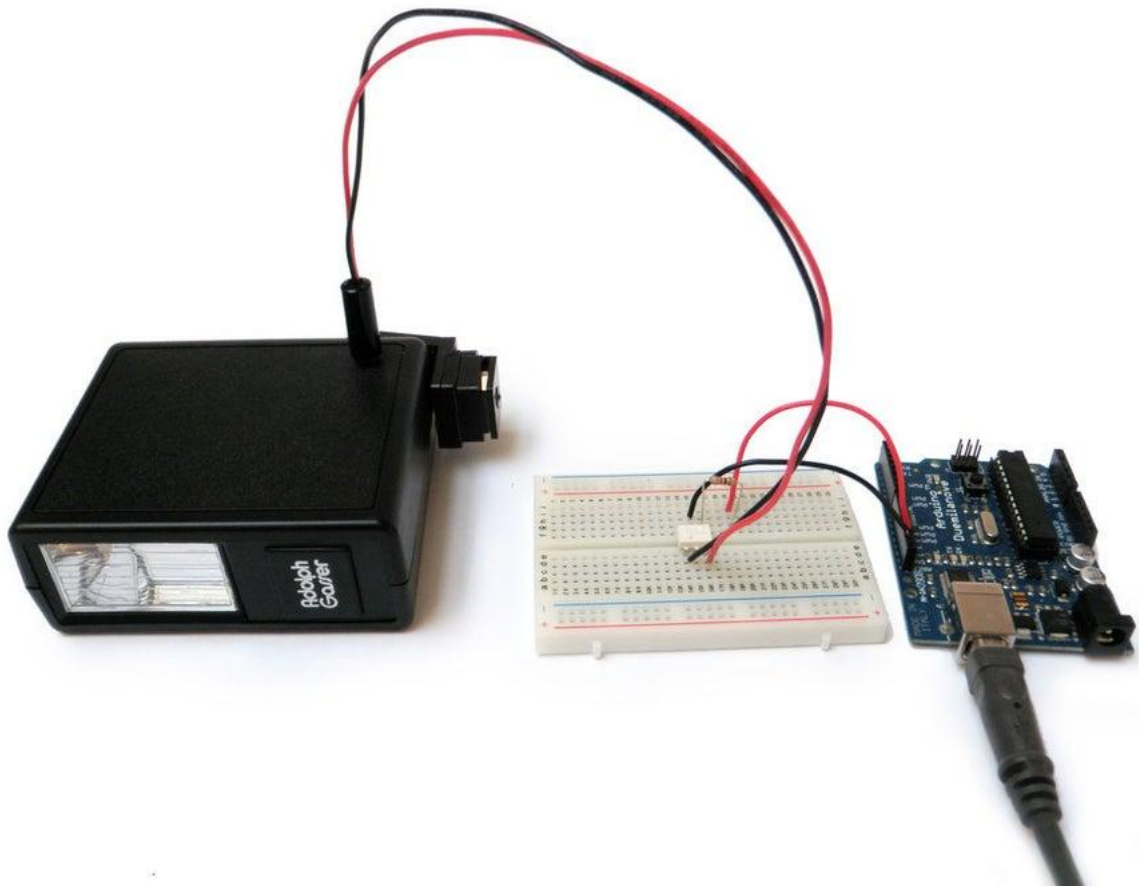
Shields are expansion adapter boards that plug in over top of the Arduino Uno and gives it special functions.

Since the Arduino is open hardware, anyone who has the inclination is free to make an Arduino shield for whatever task they wish to accomplish. On account of this, there are countless number of Arduino shields out in the wild. You can find an ever-growing list of [Arduino shields](#) in the Arduino playground. Keep in mind that there will be more shield in existence than you will find on listed on that page (as always, Google is your friend).

To give you a small sense of the capabilities of Arduino shields, check out these tutorials on how to use three official Arduino shields:

- [Wireless SD Shield](#)
- [Ethernet Shield](#)
- [Motor Shield](#)

Step 14: Building an External Circuit



As your projects get more complex, you will want to build your own circuits to interface with the Arduino. While you won't learn electronics overnight, the internet is an unbelievable resource for electronic knowledge and circuit diagrams.

Step 15: Going Beyond

