|  |  |  |
| --- | --- | --- |
| Core Functions Simple programs that demonstrate basic Arduino commands. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. 1.Basics  * [BareMinimum](http://arduino.cc/en/Tutorial/BareMinimum): The bare minimum of code needed to start an Arduino sketch. * [Blink](http://arduino.cc/en/Tutorial/Blink): Turn an LED on and off. * [DigitalReadSerial](http://arduino.cc/en/Tutorial/DigitalReadSerial): Read a switch, print the state out to the Arduino Serial Monitor. * [AnalogReadSerial](http://arduino.cc/en/Tutorial/AnalogReadSerial): Read a potentiometer, print it's state out to the Arduino Serial Monitor. * [Fade](http://arduino.cc/en/Tutorial/Fade): Demonstrates the use of analog output to fade an LED. * [ReadAnalogVoltage](http://arduino.cc/en/Tutorial/ReadAnalogVoltage) : Reads an analog input and prints the voltage to the serial monitor  2.Digital  * [Blink Without Delay](http://arduino.cc/en/Tutorial/BlinkWithoutDelay): blinking an LED without using the delay() function. * [Button](http://arduino.cc/en/Tutorial/Button): use a pushbutton to control an LED. * [Debounce](http://arduino.cc/en/Tutorial/Debounce): read a pushbutton, filtering noise. * [Button State Change](http://arduino.cc/en/Tutorial/ButtonStateChange): counting the number of button pushes. * [Input Pullup Serial](http://arduino.cc/en/Tutorial/InputPullupSerial): Demonstrates the use of INPUT\_PULLUP with pinMode(). * [Tone](http://arduino.cc/en/Tutorial/Tone): play a melody with a Piezo speaker. * [Pitch follower](http://arduino.cc/en/Tutorial/Tone2): play a pitch on a piezo speaker depending on an analog input. * [Simple Keyboard](http://arduino.cc/en/Tutorial/Tone3): a three-key musical keyboard using force sensors and a piezo speaker. * [Tone4](http://arduino.cc/en/Tutorial/Tone4): play tones on multiple speakers sequentially using the tone() command.  3.Analog  * [AnalogInOutSerial](http://arduino.cc/en/Tutorial/AnalogInOutSerial): read an analog input pin, map the result, and then use that data to dim or brighten an LED. * [Analog Input](http://arduino.cc/en/Tutorial/AnalogInput): use a potentiometer to control the blinking of an LED. * [AnalogWriteMega](http://arduino.cc/en/Tutorial/AnalogWriteMega): fade 12 LEDs on and off, one by one, using an Arduino Mega board. * [Calibration](http://arduino.cc/en/Tutorial/Calibration): define a maximum and minimum for expected analog sensor values. * [Fading](http://arduino.cc/en/Tutorial/Fading): use an analog output (PWM pin) to fade an LED. * [Smoothing](http://arduino.cc/en/Tutorial/Smoothing): smooth multiple readings of an analog input.  4.Communication These examples include code that allows the Arduino to talk to Processing sketches running on the computer. For more information or to download Processing, see [*processing.org*](http://processing.org/). There are also Max/MSP patches that can communicate with each Arduino sketch as well. For more on Max/MSP see [Cycling 74](http://www.cycling74.com). For Pd patches that can communicate with these sketches, see [Scott Fitzgerald's examples](http://www.ennuigo.com/?p=409).   * [ReadASCIIString](http://arduino.cc/en/Tutorial/ReadASCIIString): parse a comma-separated string of ints to fade an LED * [ASCII Table](http://arduino.cc/en/Tutorial/ASCIITable): demonstrates Arduino's advanced serial output functions. * [Dimmer](http://arduino.cc/en/Tutorial/Dimmer): move the mouse to change the brightness of an LED. * [Graph](http://arduino.cc/en/Tutorial/Graph): send data to the computer and graph it in Processing. * [Physical Pixel](http://arduino.cc/en/Tutorial/PhysicalPixel): turn a LED on and off by sending data to your Arduino from Processing or Max/MSP. * [Virtual Color Mixer](http://arduino.cc/en/Tutorial/VirtualColorMixer): send multiple variables from Arduino to your computer and read them in Processing or Max/MSP. * [Serial Call Response](http://arduino.cc/en/Tutorial/SerialCallResponse): send multiple vairables using a call-and-response (handshaking) method. * [Serial Call Response ASCII](http://arduino.cc/en/Tutorial/SerialCallResponseASCII): send multiple variables using a call-and-response (handshaking) method, and ASCII-encode the values before sending. * [SerialEvent](http://arduino.cc/en/Tutorial/SerialEvent): Demonstrates the use of SerialEvent(). * [Serial input (Switch (case) Statement)](http://arduino.cc/en/Tutorial/SwitchCase2): how to take different actions based on characters received by the serial port. * [MIDI](http://arduino.cc/en/Tutorial/Midi): send MIDI note messages serially. * [MultiSerialMega](http://arduino.cc/en/Tutorial/MultiSerialMega): use two of the serial ports available on the Arduino Mega.  5.Control Structures  * [If Statement](http://arduino.cc/en/Tutorial/IfStatement) (Conditional): how to use an if statement to change output conditions based on changing input conditions. * [For Loop](http://arduino.cc/en/Tutorial/ForLoop): controlling multiple LEDs with a for loop and. * [Array](http://arduino.cc/en/Tutorial/Array): a variation on the For Loop example that demonstrates how to use an array. * [While Loop](http://arduino.cc/en/Tutorial/WhileLoop): how to use a while loop to calibrate a sensor while a button is being read. * [Switch Case](http://arduino.cc/en/Tutorial/SwitchCase): how to choose between a discrete number of values. Equivalent to multiple If statements. This example shows how to divide a sensor's range into a set of four bands and to take four different actions depending on which band the result is in. * [Switch Case 2](http://arduino.cc/en/Tutorial/SwitchCase2): a second switch-case example, showing how to take different actions based in characters received in the serial port.  6.Sensors  * [ADXL3xx](http://arduino.cc/en/Tutorial/ADXL3xx): read an ADXL3xx accelerometer. * [Knock](http://arduino.cc/en/Tutorial/Knock): detect knocks with a piezo element. * [Memsic2125](http://arduino.cc/en/Tutorial/Memsic2125) : two-axis acceleromoter. * [Ping](http://arduino.cc/en/Tutorial/Ping): detecting objects with an ultrasonic range finder.  7.Display Examples of basic display control   * [LED Bar Graph](http://arduino.cc/en/Tutorial/BarGraph): how to make an LED bar graph. * [Row Column Scanning](http://arduino.cc/en/Tutorial/RowColumnScanning): how to control an 8x8 matrix of LEDs.  8.Strings  * [StringAdditionOperator](http://arduino.cc/en/Tutorial/StringAdditionOperator): add strings together in a variety of ways. * [StringAppendOperator](http://arduino.cc/en/Tutorial/StringAppendOperator): append data to strings. * [StringCaseChanges](http://arduino.cc/en/Tutorial/StringCaseChanges): change the case of a string. * [StringCharacters](http://arduino.cc/en/Tutorial/StringCharacters): get/set the value of a specific character in a string. * [StringComparisonOperators](http://arduino.cc/en/Tutorial/StringComparisonOperators): compare strings alphabetically. * [StringConstructors](http://arduino.cc/en/Tutorial/StringConstructors): how to initialize string objects. * [StringIndexOf](http://arduino.cc/en/Tutorial/StringIndexOf): look for the first/last instance of a character in a string. * [StringLength & StringLengthTrim](http://arduino.cc/en/Tutorial/StringLengthTrim): get and trim the length of a string. * [StringReplace](http://arduino.cc/en/Tutorial/StringReplace): replace individual characters in a string. * [StringStartsWithEndsWith](http://arduino.cc/en/Tutorial/StringStartsWithEndsWith): check which characters/substrings a given string starts or ends with. * [StringSubstring](http://arduino.cc/en/Tutorial/StringSubstring): look for "phrases" within a given string.  9.USB (Leonardo, Micro, and Due specific examples) The Keyboard and Mouse examples are unique to the Leonardo, Micro and Due. They demonstrate the use of libraries that are unique to the board.   * [KeyboardAndMouseControl](http://arduino.cc/en/Tutorial/KeyboardAndMouseControl): Demonstrates the Mouse and Keyboard commands in one program.  Keyboard  * [KeyboardMessage](http://arduino.cc/en/Tutorial/KeyboardMessage): Sends a text string when a button is pressed. * [KeyboardLogout](http://arduino.cc/en/Tutorial/KeyboardLogout) : Logs out the current user with key commands * [KeyboardSerial](http://arduino.cc/en/Tutorial/KeyboardSerial): Reads a byte from the serial port, and sends back a keystroke. * [KeyboardReprogram](http://arduino.cc/en/Tutorial/KeyboardReprogram) : opens a new window in the Arduino IDE and reprograms the Leonardo with a simple blink program  Mouse  * [ButtonMouseControl](http://arduino.cc/en/Tutorial/ButtonMouseControl): Control cursor movement with 5 pushbuttons. * [JoystickMouseControl](http://arduino.cc/en/Tutorial/JoystickMouseControl): Controls a computer's cursor movement with a Joystick when a button is pressed. |  | [Libraries](http://arduino.cc/en/Reference/Libraries) Examples from the libraries that are included in the Arduino software. EEPROM Library  * [EEPROM Clear](http://arduino.cc/en/Tutorial/EEPROMClear): clear the bytes in the EEPROM. * [EEPROM Read](http://arduino.cc/en/Tutorial/EEPROMRead): read the EEPROM and send its values to the computer. * [EEPROM Write](http://arduino.cc/en/Tutorial/EEPROMWrite): stores values from an analog input to the EEPROM.  Esplora Library Esplora Beginner examples   * [EsploraBlink](http://arduino.cc/en/Tutorial/EsploraBlink) : Blink the Esplora's RGB LED * [EsploraAccelerometer](http://arduino.cc/en/Tutorial/EsploraAccelerometer) : Read the values from the accelerometer * [EsploraJoystickMouse](http://arduino.cc/en/Tutorial/EsploraJoystickMouse) : Use the Esplora's joystick to control the cursor on your computer * [EsploraLedShow](http://arduino.cc/en/Tutorial/EsploraLedShow) : Use the Joystick and slider to create a light show with the LED * [EsploraLedShow2](http://arduino.cc/en/Tutorial/EsploraLedShow2) : Use the Esplora's microphone, linear potentiometer, and light sensor to change the color of the onboard LED. * [EsploraLightCalibrator](http://arduino.cc/en/Tutorial/EsploraLightCalibrator) : Read the values from the accelerometer * [EsploraMusic](http://arduino.cc/en/Tutorial/EsploraMusic) : Make some music with the Esplora * [EsploraSoundSensor](http://arduino.cc/en/Tutorial/EsploraSoundSensor) : Read the values from the Esplora's microphone * [EsploraTemperatureSensor](http://arduino.cc/en/Tutorial/EsploraTemperatureSensor) : Read the temperature sensor and get the temperature in in Farhenheit or Celsius.   Esplora Expert examples   * [EsploraKart](http://arduino.cc/en/Tutorial/EsploraKart) : Use the Esplora as a controller to play a kart racing game. * [EsploraTable](http://arduino.cc/en/Tutorial/EsploraTable) : Print the Esplora sensor information to a table format. * [EsploraRemote](http://arduino.cc/en/Tutorial/EsploraRemote) : Connect the Esplora to Processing and control the outputs. * [EsploraPong](http://arduino.cc/en/Tutorial/EsploraPong) : Play Pong with the Esplora using Processing.  Ethernet Library  * [ChatServer](http://arduino.cc/en/Tutorial/ChatServer): set up a simple chat server. * [WebClient](http://arduino.cc/en/Tutorial/WebClient): make a HTTP request. * [WebClientRepeating](http://arduino.cc/en/Tutorial/WebClientRepeating): Make repeated HTTP requests. * [WebServer](http://arduino.cc/en/Tutorial/WebServer): host a simple HTML page that displays analog sensor values. * [PachubeClient](http://arduino.cc/en/Tutorial/PachubeClient): connect to pachube.com, a free datalogging site. * [PachubeClientString](http://arduino.cc/en/Tutorial/PachubeClientString): send strings to pachube.com. * [BarometricPressureWebServer](http://arduino.cc/en/Tutorial/BarometricPressureWebServer): outputs the values from a barometric pressure sensor as a web page. * [UDPSendReceiveString](http://arduino.cc/en/Tutorial/UDPSendReceiveString): Send and receive text strings via UDP. * [UdpNtpClient](http://arduino.cc/en/Tutorial/UdpNtpClient): Query a Network Time Protocol (NTP) server using UDP. * [DnsWebClient](http://arduino.cc/en/Tutorial/DnsWebClient): DNS and DHCP-based Web client. * [DhcpChatServer](http://arduino.cc/en/Tutorial/DhcpChatServer): A simple DHCP Chat Server * [DhcpAddressPrinter](http://arduino.cc/en/Tutorial/DhcpAddressPrinter): Get an IP address via DHCP and print it out * [TwitterClient](http://arduino.cc/en/Tutorial/TwitterClient): A Twitter client with Strings * [TelnetClient](http://arduino.cc/en/Tutorial/TelnetClient): A simple Telnet client  Firmata Libraries  * [Guide to the Standard Firmata Library](http://arduino.cc/en/Reference/Firmata)  GSM Library GSM Examples   * [Make Voice Call](http://arduino.cc/en/Tutorial/GSMExamplesMakeVoiceCall): get your shield to make phone calls from the Serial Monitor * [Receive Voice Call](http://arduino.cc/en/Tutorial/GSMExamplesReceiveVoiceCall): check the status of the modem while getting voice calls * [Send SMS](http://arduino.cc/en/Tutorial/GSMExamplesSendSMS): use the Serial Monitor to type in SMS messages to different phone numbers * [Receive SMS](http://arduino.cc/en/Tutorial/GSMExamplesReceiveSMS): read SMS messages and prompt them to the Serial Monitor * [Web Client](http://arduino.cc/en/Tutorial/GSMExamplesWebClient): download the content of a website to your Arduino board through GPRS * [Web Server](http://arduino.cc/en/Tutorial/GSMExamplesWebServer): create a wireless web server through GPRS * [Twitter Client](http://arduino.cc/en/Tutorial/GSMExamplesTwitterClient): read Twitter from anywhere * [Pachube Client](http://arduino.cc/en/Tutorial/GSMExamplesPachubeClient): communicate to the Pachube sensor backbone * [Pachube Client String](http://arduino.cc/en/Tutorial/GSMExamplesPachubeClientString): communicate to the Pachube sensor backbone   GSM Tools   * [Test Modem](http://arduino.cc/en/Tutorial/GSMToolsTestModem): read the IMEI of your modem * [Test GPRS](http://arduino.cc/en/Tutorial/GSMToolsTestGPRS): test the proper functionality of the GPRS network using your SIM card * [GSM Scan Networks](http://arduino.cc/en/Tutorial/GSMToolsGsmScanNetworks): check for the available networks * [Pin Management](http://arduino.cc/en/Tutorial/GSMToolsPinManagement): manage the PIN number of your SIM card * [Band Management](http://arduino.cc/en/Tutorial/GSMToolsBandManagement): manage the band the GSM shield connects to. * [Test Web Server](http://arduino.cc/en/Tutorial/GSMToolsTestWebServer): Create a webserver with your GSM shield  LiquidCrystal Library  * [Hello World](http://arduino.cc/en/Tutorial/LiquidCrystal): displays "hello world!" and the seconds since reset. * [Blink](http://arduino.cc/en/Tutorial/LiquidCrystalBlink): control of the block-style cursor. * [Cursor](http://arduino.cc/en/Tutorial/LiquidCrystalCursor): control of the underscore-style cursor. * [Display](http://arduino.cc/en/Tutorial/LiquidCrystalDisplay): quickly blank the display without losing what's on it. * [TextDirection](http://arduino.cc/en/Tutorial/LiquidCrystalTextDirection): control which way text flows from the cursor. * [Scroll](http://arduino.cc/en/Tutorial/LiquidCrystalScroll): scroll text left and right. * [Serial input](http://arduino.cc/en/Tutorial/LiquidCrystalSerial): accepts serial input, displays it. * [SetCursor](http://arduino.cc/en/Tutorial/LiquidCrystalSetCursor): set the cursor position. * [Autoscroll](http://arduino.cc/en/Tutorial/LiquidCrystalAutoscroll): shift text right and left.  Robot Library  * [Logo](http://arduino.cc/en/Tutorial/RobotLogo) - tell your robot where to go through the on-board keyboard * [Line Following](http://arduino.cc/en/Tutorial/RobotLineFollowing) - draw a racing track and get your robot to run on it * [Disco Bot](http://arduino.cc/en/Tutorial/RobotDiscoBot) - turn your robot into an 8-bit jukebox and dance to the beat * [Compass](http://arduino.cc/en/Tutorial/RobotCompass) - plan a treasure hunt with this digital compass * [Inputs](http://arduino.cc/en/Tutorial/RobotInputs) - learn how to control the knob and the keyboard * [Wheel Calibration](http://arduino.cc/en/Tutorial/RobotWheelCalibration) - tune the wheels to perform even better * [Runaway Robot](http://arduino.cc/en/Tutorial/RobotRunawayRobot) - play tag with your robot using a distance sensor * [Remote control](http://arduino.cc/en/Tutorial/RobotRemoteControl) - reuse that old tv-remote to command the bot on distance * [Picture browser](http://arduino.cc/en/Tutorial/RobotPictureBrowser) - want to use your own images? This is how * [Rescue](http://arduino.cc/en/Tutorial/RobotRescue) - train your robot to look for hidden pearls in a maze * [Hello User](http://arduino.cc/en/Tutorial/RobotHelloUser) - hack the robot's welcome demo and make your own  SPI Library  * [BarometricPressureSensor](http://arduino.cc/en/Tutorial/BarometricPressureSensor): read air pressure and temperature from a sensor using the SPI protocol. * [SPIDigitalPot](http://arduino.cc/en/Tutorial/SPIDigitalPot): control a AD5206 digital potentiometer using the SPI protocol.  Servo Library  * [Knob](http://arduino.cc/en/Tutorial/Knob): control the shaft of a servo motor by turning a potentiometer. * [Sweep](http://arduino.cc/en/Tutorial/Sweep): sweeps the shaft of a servo motor back and forth.  Software Serial Library  * [Software Serial Example](http://arduino.cc/en/Tutorial/SoftwareSerialExample): how to use the SoftwareSerial Library...Because sometimes one serial port just isn't enough! * [Two Port Receive](http://arduino.cc/en/Tutorial/TwoPortReceive): how to work with multiple software serial ports.  Stepper Library  * [Motor Knob](http://arduino.cc/en/Tutorial/MotorKnob): control a highly accurate stepper motor using a potentiometer.  TFT Library Esplora   * [Esplora TFT Bitmap Logo](http://arduino.cc/en/Tutorial/EsploraTFTBitmapLogo): Read an image file from a micro-SD card and draw it at random locations. * [Esplora TFT Color Picker](http://arduino.cc/en/Tutorial/EsploraTFTColorPicker): Using the joystick and slider, change the color of the TFT screen * [Esplora TFT Etch a Sketch](http://arduino.cc/en/Tutorial/EsploraTFTEtchASketch): An Esplora implementation of the classic Etch-a-Sketch * [Esplora TFT Graph](http://arduino.cc/en/Tutorial/EsploraTFTGraph): Graph the values from the light sensor to the TFT * [Esplora TFT Horizon](http://arduino.cc/en/Tutorial/EsploraTFTHorizon): Draw an artificial horizon line based on the tilt from the accelerometer * [Esplora TFT Pong](http://arduino.cc/en/Tutorial/EsploraTFTPong): A basic implementation of the classic game * [Esplora TFT Temperature](http://arduino.cc/en/Tutorial/EsploraTFTTemp): Check the temperature with the onboard sensor and display it on screen   Arduino   * [TFT Bitmap Logo](http://arduino.cc/en/Tutorial/TFTBitmapLogo): Read an image file from a micro-SD card and draw it at random locations. * [TFT Display Text](http://arduino.cc/en/Tutorial/TFTDisplayText) : Read the value of a sensor and print it on the screen. * [TFT Pong](http://arduino.cc/en/Tutorial/TFTPong): An Arduino implementation of the classic game * [Etch a Sketch](http://arduino.cc/en/Tutorial/TFTEtchASketch): An Arduino version of the classic Etch-a-Sketch * [Color Picker](http://arduino.cc/en/Tutorial/TFTColorPicker): With three sensors, change the color of the TFT screen * [Graph](http://arduino.cc/en/Tutorial/TFTGraph): Graph the values from a variable resistor to the TFT  Wire Library  * [SFRRanger\_reader](http://arduino.cc/en/Tutorial/SFRRangerReader): read a Devantech SRFxx ultra-sonic range finder using I2C communication. * [digital\_potentiometer](http://arduino.cc/en/Tutorial/DigitalPotentiometer): control a AD5171 digital pot using the Wire Library. * [master reader/slave sender](http://arduino.cc/en/Tutorial/MasterReader): set up two (or more) arduino boards to share information via a master reader/slave sender configuration. * [master writer/slave reader](http://arduino.cc/en/Tutorial/MasterWriter): allow two (or more) arduino boards to share information using a master writer/slave reader set up.  WiFi Library  * [ConnectNoEncryption](http://arduino.cc/en/Tutorial/ConnectNoEncryption) : Demonstrates how to connect to an open network * [ConnectWithWEP](http://arduino.cc/en/Tutorial/ConnectWithWEP) : Demonstrates how to connect to a network that is encrypted with WEP * [ConnectWithWPA](http://arduino.cc/en/Tutorial/ConnectWithWPA) : Demonstrates how to connect to a network that is encrypted with WPA2 Personal * [ScanNetworks](http://arduino.cc/en/Tutorial/ScanNetworks) : Displays all WiFi networks in range * [WiFiChatServer](http://arduino.cc/en/Tutorial/WiFiChatServer) : Set up a simple chat server * [WiFiPachubeClient](http://arduino.cc/en/Tutorial/WiFiPachubeClient) : connect to pachube.com, a free datalogging site * [WiFiPachubeClientString](http://arduino.cc/en/Tutorial/WiFiPachubeClientString): send strings to pachube.com * [WiFiTwitterClient](http://arduino.cc/en/Tutorial/WiFiTwitterClient) : A Twitter client with Strings * [WiFiWebClient](http://arduino.cc/en/Tutorial/WiFiWebClient) : Connect to a remote webserver * [WiFiWebClientRepeating](http://arduino.cc/en/Tutorial/WiFiWebClientRepeating): Repeatedly make HTTP calls to a server * [WiFiWebServer](http://arduino.cc/en/Tutorial/WiFiWebServer) : Serve a webpage from the WiFi shield  Arduino as ISP Programmer [ArduinoISP](http://arduino.cc/en/Tutorial/ArduinoISP) turns your Arduino into an in-circuit programmer to re-program Atmega chips. Useful when you need to re-load the bootloader on an Arduino, if you're [going from Arduino to an Atmega on a breadboard](http://arduino.cc/en/Tutorial/ArduinoToBreadboard), or if you're making your own [Arduino-compatible circuit on a breadboard](http://arduino.cc/en/Main/Standalone). |

Programms - Arduino

**Blink**

This example shows the simplest thing you can do with an Arduino to see physical output: it blinks an LED.

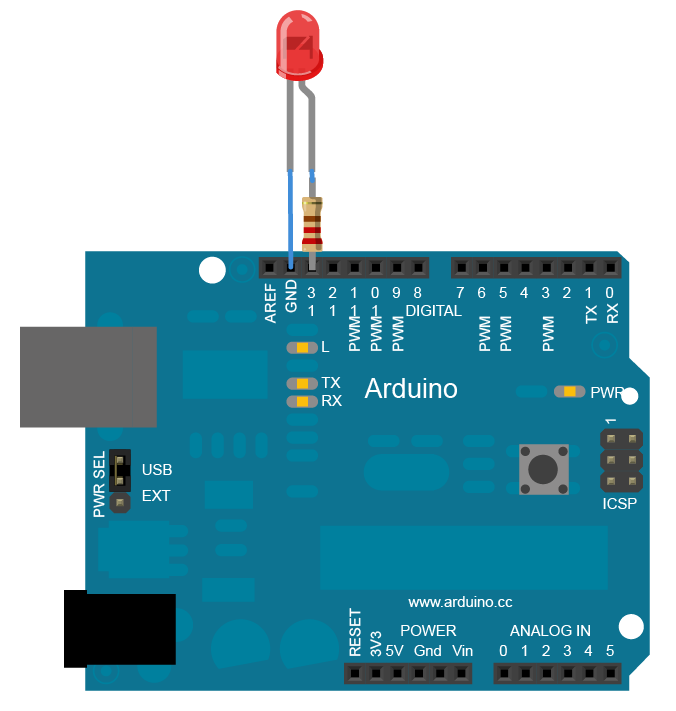
**Hardware Required**

* Arduino Board
* LED

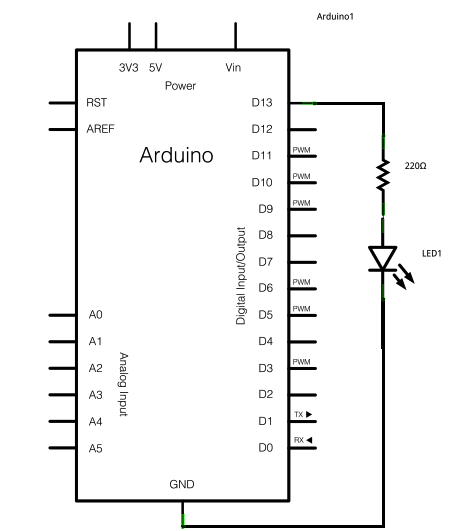
**Circuit**

To build the circuit, attach a 220-ohm resistor to pin 13. Then attach the long leg of an LED (the positive leg, called the anode) to the resistor. Attach the short leg (the negative leg, called the cathode) to ground. Then plug your Arduino board into your computer, start the Arduino program, and enter the code below.

Most Arduino boards already have an LED attached to pin 13 on the board itself. If you run this example with no hardware attached, you should see that LED blink.

[](http://arduino.cc/en/uploads/Tutorial/ExampleCircuit_bb.png)

**Schematic**

[](http://arduino.cc/en/uploads/Tutorial/ExampleCircuit_sch.png)

**Code**

In the program below, the first thing you do is to initialize pin 13 as an output pin with the line

pinMode(13, OUTPUT);

In the main loop, you turn the LED on with the line:

digitalWrite(13, HIGH);

This supplies 5 volts to pin 13. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:

digitalWrite(13, LOW);

That takes pin 13 back to 0 volts, and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the delay() commands tell the Arduino to do nothing for 1000 milliseconds, or one second. When you use the delay() command, nothing else happens for that amount of time. Once you've understood the basic examples, check out the [BlinkWithoutDelay](http://arduino.cc/en/Tutorial/BlinkWithoutDelay) example to learn how to create a delay while doing other things.

Once you've understood this example, check out the [DigitalReadSerial](http://arduino.cc/en/Tutorial/DigitalReadSerial) example to learn how read a switch connected to the Arduino.

/\*  
  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  
}

## setup()

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

### Example

int buttonPin = 3;

void setup()

{

Serial.begin(9600);

pinMode(buttonPin, INPUT);

}

void loop()

{

// ...

}

# loop()

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

### Example

const int buttonPin = 3;

// setup initializes serial and the button pin

void setup()

{

Serial.begin(9600);

pinMode(buttonPin, INPUT);

}

// loop checks the button pin each time,

// and will send serial if it is pressed

void loop()

{

if (digitalRead(buttonPin) == HIGH)

Serial.write('H');

else

Serial.write('L');

delay(1000);

}

## pinMode()

#### Description

Configures the specified pin to behave either as an input or an output. See the description of [digital pins](http://arduino.cc/en/Tutorial/DigitalPins) for details on the functionality of the pins.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT\_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.

#### Syntax

pinMode(pin, mode)

#### Parameters

pin: the number of the pin whose mode you wish to set

mode: [INPUT](http://arduino.cc/en/Reference/Constants), [OUTPUT](http://arduino.cc/en/Reference/Constants), or [INPUT\_PULLUP](http://arduino.cc/en/Reference/Constants). (see the [digital pins](http://arduino.cc/en/Tutorial/DigitalPins) page for a more complete description of the functionality.)

#### Returns

None

#### Example

int ledPin = 13;                 // LED connected to digital pin 13  
  
void setup()  
{  
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output  
}  
  
void loop()  
{  
  digitalWrite(ledPin, HIGH);   // sets the LED on  
  delay(1000);                  // waits for a second  
  digitalWrite(ledPin, LOW);    // sets the LED off  
  delay(1000);                  // waits for a second  
}

## digitalWrite()

#### Description

Write a [HIGH](http://arduino.cc/en/Reference/Constants) or a [LOW](http://arduino.cc/en/Reference/Constants) value to a digital pin.

If the pin has been configured as an OUTPUT with [pinMode](http://arduino.cc/en/Reference/PinMode)(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

If the pin is configured as an INPUT, writing a HIGH value with digitalWrite() will enable an internal 20K pullup resistor (see the [tutorial on digital pins](http://arduino.cc/en/Tutorial/DigitalPins)). Writing LOW will disable the pullup. The pullup resistor is enough to light an LED dimly, so if LEDs appear to work, but very dimly, this is a likely cause. The remedy is to set the pin to an output with the pinMode() function.

**NOTE:** Digital pin 13 is harder to use as a digital input than the other digital pins because it has an LED and resistor attached to it that's soldered to the board on most boards. If you enable its internal 20k pull-up resistor, it will hang at around 1.7 V instead of the expected 5V because the onboard LED and series resistor pull the voltage level down, meaning it always returns LOW. If you must use pin 13 as a digital input, use an external pull down resistor.

#### Syntax

digitalWrite(pin, value)

#### Parameters

pin: the pin number

value: [HIGH](http://arduino.cc/en/Reference/Constants) or [LOW](http://arduino.cc/en/Reference/Constants)

#### Returns

none

#### Example

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

void setup()

{

pinMode(ledPin, OUTPUT); // sets the digital pin as output

}

void loop()

{

digitalWrite(ledPin, HIGH); // sets the LED on

delay(1000); // waits for a second

digitalWrite(ledPin, LOW); // sets the LED off

delay(1000); // waits for a second

}

Sets pin 13 to HIGH, makes a one-second-long delay, and sets the pin back to LOW.

#### Note

The analog input pins can be used as digital pins, referred to as A0, A1, etc.

## delay()

#### Description

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

#### Syntax

delay(ms)

#### Parameters

ms: the number of milliseconds to pause (unsigned long)

#### Returns

nothing

#### Example

int ledPin = 13;                 // LED connected to digital pin 13  
  
void setup()  
{  
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output  
}  
  
void loop()  
{  
  digitalWrite(ledPin, HIGH);   // sets the LED on  
  delay(1000);                  // waits for a second  
  digitalWrite(ledPin, LOW);    // sets the LED off  
  delay(1000);                  // waits for a second  
}

#### Caveat

While it is easy to create a blinking LED with the delay() function, and many sketches use short delays for such tasks as switch debouncing, the use of delay() in a sketch has significant drawbacks. No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function, so in effect, it brings most other activity to a halt. For alternative approaches to controlling timing see the [millis()](http://arduino.cc/en/Reference/Millis) function and the sketch sited below. More knowledgeable programmers usually avoid the use of delay() for timing of events longer than 10's of milliseconds unless the Arduino sketch is very simple.

Certain things do go on while the delay() function is controlling the Atmega chip however, because the delay function does not disable interrupts. Serial communication that appears at the RX pin is recorded, PWM ([analogWrite](http://arduino.cc/en/Reference/AnalogWrite)) values and pin states are maintained, and [interrupts](http://arduino.cc/en/Reference/AttachInterrupt) will work as they should.

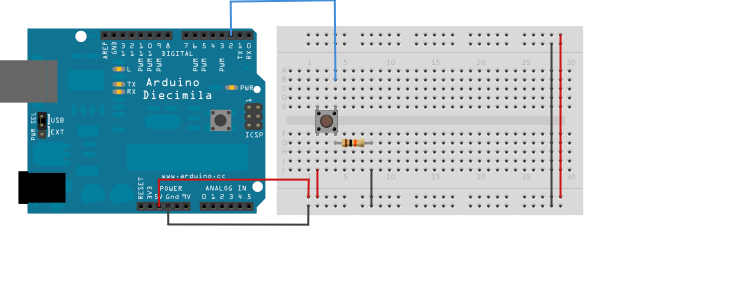
**Digital Read Serial**

This example shows you how to monitor the state of a switch by establishing [serial communication](http://arduino.cc/en/Reference/Serial) between your Arduino and your computer over USB.

**Hardware Required**

* Arduino Board
* A momentary switch, button, or toggle switch
* 10k ohm resistor
* breadboard
* hook-up wire

**Circuit**

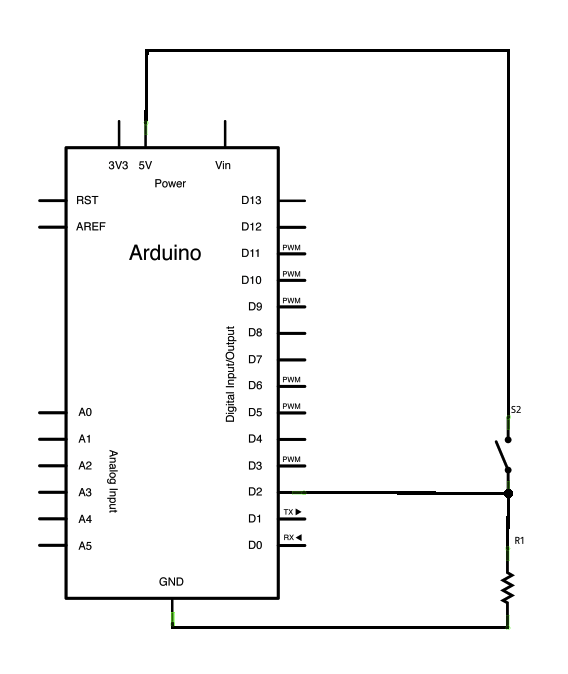


Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 KOhms) to ground. The other leg of the button connects to the 5 volt supply.

Pushbuttons or switches connect two points in a circuit when you press them. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and reads as LOW, or 0. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that the pin reads as HIGH, or 1.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it doesn't have a solid connection to voltage or ground, and it will randomly return either HIGH or LOW. That's why you need a pull-down resistor in the circuit.

**Schematic**

[](http://arduino.cc/en/uploads/Tutorial/button_sch.png)

**Code**

In the program below, the very first thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the line:

Serial.begin(9600);

Next, initialize digital pin 2, the pin that will read the output from your button, as an input:

pinMode(2,INPUT);

Now that your setup has been completed, move into the main loop of your code. When your button is pressed, 5 volts will freely flow through your circuit, and when it is not pressed, the input pin will be connected to ground through the 10-kilohm resistor. This is a digital input, meaning that the switch can only be in either an on state (seen by your Arduino as a "1", or HIGH) or an off state (seen by your Arduino as a "0", or LOW), with nothing in between.

The first thing you need to do in the main loop of your program is to establish a variable to hold the information coming in from your switch. Since the information coming in from the switch will be either a "1" or a "0", you can use an [int datatype](http://arduino.cc/en/Reference/Int). Call this variable sensorValue, and set it to equal whatever is being read on digital pin 2. You can accomplish all this with just one line of code:

int sensorValue = digitalRead(2);

Once the Arduino has read the input, make it print this information back to the computer as a decimal value. You can do this with the command [Serial.println](http://arduino.cc/en/Serial/Println)() in our last line of code:

Serial.println(sensorValue);

Now, when you open your Serial Monitor in the Arduino environment, you will see a stream of "0"s if your switch is open, or "1"s if your switch is closed.

/\*  
  DigitalReadSerial  
 Reads a digital input on pin 2, prints the result to the serial monitor   
   
 This example code is in the public domain.  
 \*/  
  
// digital pin 2 has a pushbutton attached to it. Give it a name:  
int pushButton = 2;  
  
// the setup routine runs once when you press reset:  
void setup() {  
  // initialize serial communication at 9600 bits per second:  
  Serial.begin(9600);  
  // make the pushbutton's pin an input:  
  pinMode(pushButton, INPUT);  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
  // read the input pin:  
  int buttonState = digitalRead(pushButton);  
  // print out the state of the button:  
  Serial.println(buttonState);  
  delay(1);        // delay in between reads for stability  
}

**Analog Read Serial**

This example shows you how to read analog input from the physical world using a potentiometer. A **potentiometer** is a simple mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input on your Arduino, it is possible to measure the amount of resistance produced by a potentiometer (or *pot* for short) as an analog value. In this example you will monitor the state of your potentiometer after establishing serial communication between your Arduino and your computer.

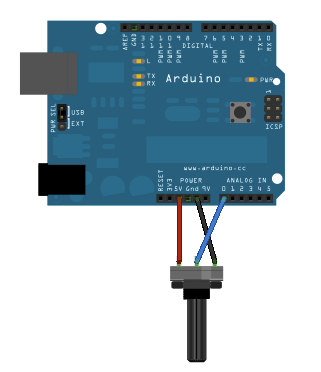
**Hardware Required**

* Arduino Board
* 10-kilohm Potentiometer

**Circuit**

Connect the three wires from the potentiometer to your Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 0 to the middle pin of the potentiometer.

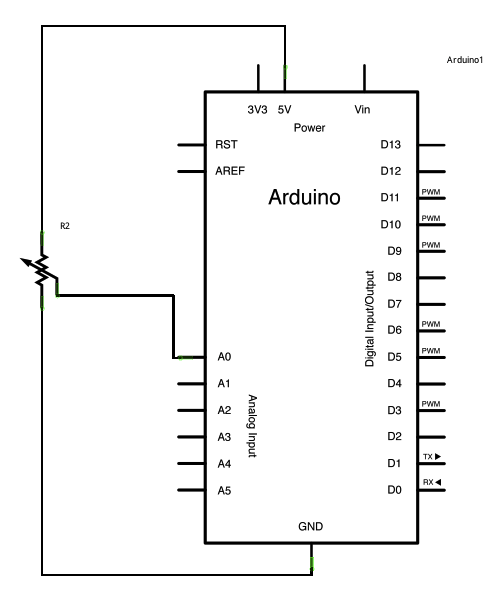
click the image to enlarge

[](http://arduino.cc/en/uploads/Tutorial/AnalogReadSerial_BB.png)

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 kilohms), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the **analog voltage** that you're reading as an input.

The Arduino has a circuit inside called an **analog-to-digital converter** that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, [analogRead](http://arduino.cc/en/Reference/AnalogRead)() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

**Schematic**

[](http://arduino.cc/en/uploads/Tutorial/AnalogReadSerial_sch.png)

**Code**

In the program below, the only thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the command:

Serial.begin(9600);

Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an [int datatype](http://arduino.cc/en/Reference/Int)) coming in from your potentiometer:

int sensorValue = analogRead(A0);

Finally, you need to print this information to your serial window as a decimal (DEC) value. You can do this with the command [Serial.println](http://arduino.cc/en/Serial/Println)() in your last line of code:

Serial.println(sensorValue, DEC)

Now, when you open your Serial Monitor in the Arduino development environment (by clicking the button directly to the right of the "Upload" button in the header of the program), you should see a steady stream of numbers ranging from 0-1023, correlating to the position of the pot. As you turn your potentiometer, these numbers will respond almost instantly.

/\*  
  AnalogReadSerial  
  Reads an analog input on pin 0, prints the result to the serial monitor.  
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.  
   
 This example code is in the public domain.  
 \*/  
  
// the setup routine runs once when you press reset:  
void setup() {  
  // initialize serial communication at 9600 bits per second:  
  Serial.begin(9600);  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
  // read the input on analog pin 0:  
  int sensorValue = analogRead(A0);  
  // print out the value you read:  
  Serial.println(sensorValue);  
  delay(1);        // delay in between reads for stability  
}

Top of Form

## int

#### Description

Integers are your primary data-type for number storage.

On the Arduino Uno (and other ATMega based boards) an int stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).   
On the Arduino Due, an int stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

int's store negative numbers with a technique called [2's complement math.](http://en.wikipedia.org/wiki/2%27s_complement) The highest bit, sometimes referred to as the "sign" bit, flags the number as a negative number. The rest of the bits are inverted and 1 is added.

The Arduino takes care of dealing with negative numbers for you, so that arithmetic operations work transparently in the expected manner. There can be an unexpected complication in dealing with the [bitshift right operator (>>)](http://arduino.cc/en/Reference/Bitshift) however.

#### Example

int ledPin = 13;

#### Syntax

int var = val;

* var - your int variable name
* val - the value you assign to that variable

#### Coding Tip

When variables are made to exceed their maximum capacity they "roll over" back to their minimum capacity, note that this happens in both directions. Example for a 16-bit int:

int x;

x = -32768;

x = x - 1; // x now contains 32,767 - rolls over in neg. direction

x = 32767;

x = x + 1; // x now contains -32,768 - rolls over

## analogRead()

#### Description

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using [analogReference](http://arduino.cc/en/Reference/AnalogReference)().

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

#### Syntax

analogRead(pin)

#### Parameters

pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

#### Returns

int (0 to 1023)

#### Note

If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

#### Example

int analogPin = 3; // potentiometer wiper (middle terminal) connected to analog pin 3

// outside leads to ground and +5V

int val = 0; // variable to store the value read

void setup()

{

Serial.begin(9600); // setup serial

}

void loop()

{

val = analogRead(analogPin); // read the input pin

Serial.println(val); // debug value

}

## println()

#### Description

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as [Serial.print](http://arduino.cc/en/Serial/Print)().

#### Syntax

Serial.println(val)   
Serial.println(val, format)

#### Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

#### Returns

size\_t (long): println() returns the number of bytes written, though reading that number is optional

#### Example:

/\*  
  Analog input  
  
 reads an analog input on analog in 0, prints the value out.  
  
 created 24 March 2006  
 by Tom Igoe  
 \*/  
  
int analogValue = 0;    // variable to hold the analog value  
  
void setup() {  
  // open the serial port at 9600 bps:  
  Serial.begin(9600);  
}  
  
void loop() {  
  // read the analog input on pin 0:  
  analogValue = analogRead(0);  
  
  // print it out in many formats:  
  Serial.println(analogValue);       // print as an ASCII-encoded decimal  
  Serial.println(analogValue, DEC);  // print as an ASCII-encoded decimal  
  Serial.println(analogValue, HEX);  // print as an ASCII-encoded hexadecimal  
  Serial.println(analogValue, OCT);  // print as an ASCII-encoded octal  
  Serial.println(analogValue, BIN);  // print as an ASCII-encoded binary  
  
  // delay 10 milliseconds before the next reading:  
  delay(10);  
}

**Fading**

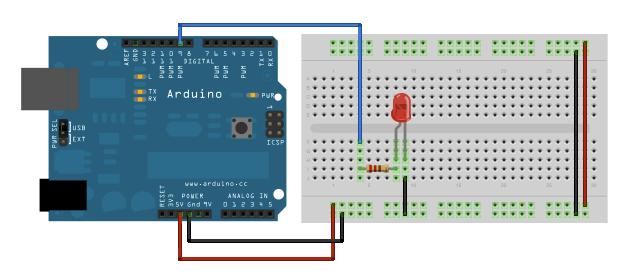
Demonstrates the use of the [analogWrite()](http://arduino.cc/en/Reference/AnalogWrite) function in fading an LED off and on. AnalogWrite uses [pulse width modulation (PWM)](http://arduino.cc/en/Tutorial/PWM), turning a digital pin on and off very quickly, to create a fading effect.

**Hardware Required**

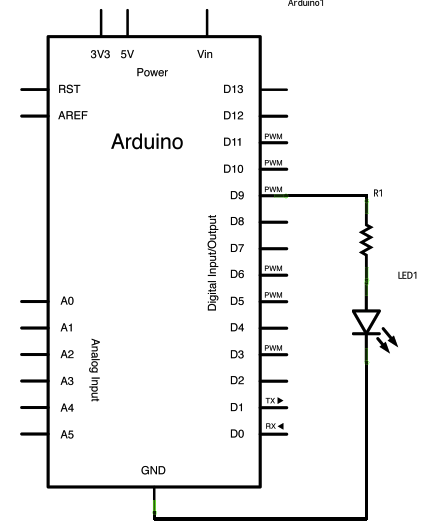
* Arduino board
* Breadboard
* a LED
* a 220 ohm resistor

**Circuit**

Connect the **anode** (the longer, positive leg) of your LED to digital output pin 9 on your Arduino through a 220-ohm resistor. Connect the **cathode** (the shorter, negative leg) directly to ground.

[](http://arduino.cc/en/uploads/Tutorial/simplefade_bb.png)

**Schematic**

[](http://arduino.cc/en/uploads/Tutorial/simplefade_pin9_schem.png)

**Code**

After declaring pin 9 to be your ledPin, there is nothing to do in the setup() function of your code.

The analogWrite() function that you will be using in the main loop of your code requires two arguments: One telling the function which pin to write to, and one indicating what [PWM](http://arduino.cc/en/Tutorial/PWM) value to write.

In order to fade your LED off and on, gradually increase your PWM value from 0 (all the way off) to 255 (all the way on), and then back to 0 once again to complete the cycle. In the sketch below, the PWM value is set using a variable called brightness. Each time through the loop, it increases by the value of the variable fadeAmount.

If brightness is at either extreme of its value (either 0 or 255), then fadeAmount is changed to its negative. In other words, if fadeAmount is 5, then it is set to -5. If it's 55, then it's set to 5. The next time through the loop, this change causes brightness to change direction as well.

analogWrite() can change the PWM value very fast, so the delay at the end of the sketch controls the speed of the fade. Try changing the value of the delay and see how it changes the program.

/\*  
 Fade  
   
 This example shows how to fade an LED on pin 9  
 using the analogWrite() function.  
   
 This example code is in the public domain.  
 \*/  
  
int led = 9;           // the pin that the LED is attached to  
int brightness = 0;    // how bright the LED is  
int fadeAmount = 5;    // how many points to fade the LED by  
  
// the setup routine runs once when you press reset:  
void setup()  {   
  // declare pin 9 to be an output:  
  pinMode(led, OUTPUT);  
}   
  
// the loop routine runs over and over again forever:  
void loop()  {   
  // set the brightness of pin 9:  
  analogWrite(led, brightness);      
  
  // change the brightness for next time through the loop:  
  brightness = brightness + fadeAmount;  
  
  // reverse the direction of the fading at the ends of the fade:   
  if (brightness == 0 || brightness == 255) {  
    fadeAmount = -fadeAmount ;   
  }       
  // wait for 30 milliseconds to see the dimming effect      
  delay(30);                              
}

## PWM

The Fading example demonstrates the use of analog output (PWM) to fade an LED. It is available in the File->Sketchbook->Examples->Analog menu of the Arduino software.

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to [analogWrite](http://arduino.cc/en/Reference/AnalogWrite)() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.



Once you get this example running, grab your arduino and shake it back and forth. What you are doing here is essentially mapping time across the space. To our eyes, the movement blurs each LED blink into a line. As the LED fades in and out, those little lines will grow and shrink in length. Now you are seeing the pulse width.

## for statements

#### Desciption

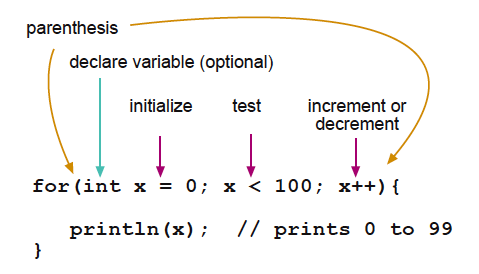
The **for** statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The **for** statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the **for** loop header:

**for** (**initialization**; **condition**; **increment**) {

//statement(s);

}



The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.

#### Example

// Dim an LED using a PWM pin

int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup()

{

// no setup needed

}

void loop()

{

for (int i=0; i <= 255; i++){

analogWrite(PWMpin, i);

delay(10);

}

}

#### Coding Tips

The C **for** loop is much more flexible than **for** loops found in some other computer languages, including BASIC. Any or all of the three header elements may be omitted, although the semicolons are required. Also the statements for initialization, condition, and increment can be any valid C statements with unrelated variables, and use any C datatypes including floats. These types of unusual **for** statements may provide solutions to some rare programming problems.

For example, using a multiplication in the increment line will generate a logarithmic progression:

for(int x = 2; x < 100; x = x \* 1.5){

println(x);

}

Generates: 2,3,4,6,9,13,19,28,42,63,94

Another example, fade an LED up and down with one **for** loop:

void loop()

{

int x = 1;

for (int i = 0; i > -1; i = i + x){

analogWrite(PWMpin, i);

if (i == 255) x = -1; // switch direction at peak

delay(10);

}

}

## while loops

#### Description

**while** loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the **while** loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

#### Syntax

while(expression){

// statement(s)

}

#### Parameters

expression - a (boolean) C statement that evaluates to true or false

#### Example

var = 0;  
while(var < 200){  
  // do something repetitive 200 times  
  var++;  
}

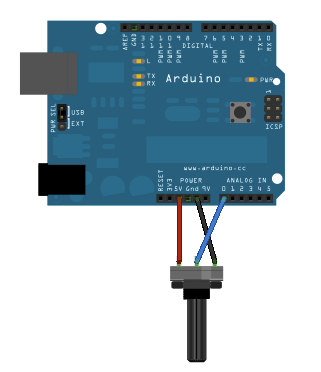
**Analog Read Voltage**

This example shows you how to read an analog input on Pin 0, convert the values from analogRead() into voltage, and print it out to the serial monitor.

**Hardware Required**

* Arduino Board
* a variable resistor, like a potentiometer

**Circuit**

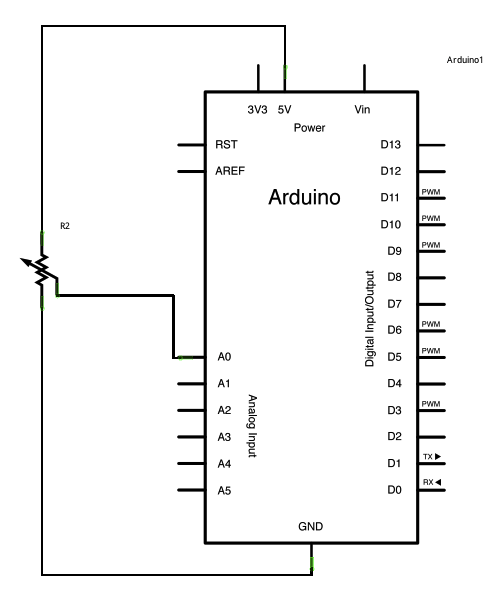


Connect the three wires from the potentiometer to your Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 kilohms), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the **analog voltage** that you're reading as an input.

The Arduino has a circuit inside called an **analog-to-digital converter** that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, [analogRead](http://arduino.cc/en/Reference/AnalogRead)() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

**Schematic**

[](http://arduino.cc/en/uploads/Tutorial/AnalogReadSerial_sch.png)

**Code**

In the program below, the very first thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the line:

Serial.begin(9600);

Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an [int datatype](http://arduino.cc/en/Reference/Int)) coming in from your potentiometer:

int sensorValue = analogRead(A0);

To change the values from 0-1023 to a range that corresponds to the voltage the pin is reading, you'll need to create another variable, a [float](http://arduino.cc/en/Reference/Float), and do a little math. To scale the numbers between 0.0 and 5.0, divide 5.0 by 1023.0 and multiply that by **sensorValue** :

float voltage= sensorValue \* (5.0 / 1024.0);

Finally, you need to print this information to your serial window as. You can do this with the command [Serial.println](http://arduino.cc/en/Serial/Println)() in your last line of code:

Serial.println(voltage)

Now, when you open your Serial Monitor in the Arduino development environment (by clicking the button directly to the right of the "Upload" button in the header of the program), you should see a steady stream of numbers ranging from 0.0 - 5.0. As you turn the pot, the values will change, corresponding to the voltage coming into pin A0.

/\*  
  ReadAnalogVoltage  
  Reads an analog input on pin 0, converts it to voltage, and prints the result to the serial monitor.  
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.  
   
 This example code is in the public domain.  
 \*/  
  
// the setup routine runs once when you press reset:  
void setup() {  
  // initialize serial communication at 9600 bits per second:  
  Serial.begin(9600);  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
  // read the input on analog pin 0:  
  int sensorValue = analogRead(A0);  
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):  
  float voltage = sensorValue \* (5.0 / 1023.0);  
  // print out the value you read:  
  Serial.println(voltage);  
}

## float

#### Description

Datatype for floating-point numbers, a number that has a decimal point. Floating-point numbers are often used to approximate analog and continuous values because they have greater resolution than integers. Floating-point numbers can be as large as 3.4028235E+38 and as low as -3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

Floats have only 6-7 decimal digits of precision. That means the total number of digits, not the number to the right of the decimal point. Unlike other platforms, where you can get more precision by using a double (e.g. up to 15 digits), on the Arduino, double is the same size as float.

Floating point numbers are not exact, and may yield strange results when compared. For example 6.0 / 3.0 may not equal 2.0. You should instead check that the absolute value of the difference between the numbers is less than some small number.

Floating point math is also much slower than integer math in performing calculations, so should be avoided if, for example, a loop has to run at top speed for a critical timing function. Programmers often go to some lengths to convert floating point calculations to integer math to increase speed.

#### Examples

float myfloat;

float sensorCalbrate = 1.117;

#### Syntax

float var = val;

* var - your float variable name
* val - the value you assign to that variable

#### Example Code

int x;

int y;

float z;

x = 1;

y = x / 2; // y now contains 0, ints can't hold fractions

z = (float)x / 2.0; // z now contains .5 (you have to use 2.0, not 2)

## Serial

|  |  |  |
| --- | --- | --- |
| Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): **Serial**. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.  You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().  The [Arduino Mega](http://arduino.cc/en/Main/ArduinoBoardMega2560) has three additional serial ports: **Serial1** on pins 19 (RX) and 18 (TX), **Serial2** on pins 17 (RX) and 16 (TX), **Serial3** on pins 15 (RX) and 14 (TX). To use these pins to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the Mega's USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your Mega to your device's ground. (Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.)  The [Arduino Due](http://arduino.cc/en/Main/ArduinoBoardDue) has three additional 3.3V TTL serial ports: **Serial1** on pins 19 (RX) and 18 (TX); **Serial2** on pins 17 (RX) and 16 (TX), **Serial3** on pins 15 (RX) and 14 (TX). Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip, which is connected to the USB debug port. Additionally, there is a native USB-serial port on the SAM3X chip, *SerialUSB*'.  The Arduino Leonardo board uses **Serial1** to communicate via TTL (5V) serial on pins 0 (RX) and 1 (TX). **Serial** is reserved for USB CDC communication. For more information, refer to the Leonardo [getting started](http://arduino.cc/en/Guide/ArduinoLeonardo) page and [hardware page](http://arduino.cc/en/Main/ArduinoBoardLeonardo). |  | Functions  * [if (Serial)](http://arduino.cc/en/Serial/IfSerial) * [available](http://arduino.cc/en/Serial/Available)() * [begin](http://arduino.cc/en/Serial/Begin)() * [end](http://arduino.cc/en/Serial/End)() * [find](http://arduino.cc/en/Serial/Find)() * [findUntil](http://arduino.cc/en/Serial/FindUntil)() * [flush](http://arduino.cc/en/Serial/Flush)() * [parseFloat](http://arduino.cc/en/Serial/ParseFloat)() * [parseInt](http://arduino.cc/en/Serial/ParseInt)() * [peek](http://arduino.cc/en/Serial/Peek)() * [print](http://arduino.cc/en/Serial/Print)() * [println](http://arduino.cc/en/Serial/Println)() * [read](http://arduino.cc/en/Serial/Read)() * [readBytes](http://arduino.cc/en/Serial/ReadBytes)() * [readBytesUntil](http://arduino.cc/en/Serial/ReadBytesUntil)() * [setTimeout](http://arduino.cc/en/Serial/SetTimeout)() * [write](http://arduino.cc/en/Serial/Write)() * [serialEvent](http://arduino.cc/en/Reference/SerialEvent)()  Examples  * [ReadASCIIString](http://arduino.cc/en/Tutorial/ReadASCIIString) * [ASCII Table](http://arduino.cc/en/Tutorial/ASCIITable) * [Dimmer](http://arduino.cc/en/Tutorial/Dimmer) * [Graph](http://arduino.cc/en/Tutorial/Graph) * [Physical Pixel](http://arduino.cc/en/Tutorial/PhysicalPixel) * [Virtual Color Mixer](http://arduino.cc/en/Tutorial/VirtualColorMixer) * [Serial Call Response](http://arduino.cc/en/Tutorial/SerialCallResponse) * [Serial Call Response ASCII](http://arduino.cc/en/Tutorial/SerialCallResponseASCII) |

[Arduino](http://arduino.cc/en)

Top of Form

## int

#### Description

Integers are your primary data-type for number storage.

On the Arduino Uno (and other ATMega based boards) an int stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).   
On the Arduino Due, an int stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

int's store negative numbers with a technique called [2's complement math.](http://en.wikipedia.org/wiki/2%27s_complement) The highest bit, sometimes referred to as the "sign" bit, flags the number as a negative number. The rest of the bits are inverted and 1 is added.

The Arduino takes care of dealing with negative numbers for you, so that arithmetic operations work transparently in the expected manner. There can be an unexpected complication in dealing with the [bitshift right operator (>>)](http://arduino.cc/en/Reference/Bitshift) however.

#### Example

int ledPin = 13;

#### Syntax

int var = val;

* var - your int variable name
* val - the value you assign to that variable

#### Coding Tip

When variables are made to exceed their maximum capacity they "roll over" back to their minimum capacity, note that this happens in both directions. Example for a 16-bit int:

int x;

x = -32768;

x = x - 1; // x now contains 32,767 - rolls over in neg. direction

x = 32767;

x = x + 1; // x now contains -32,768 - rolls over

## Variables

A variable is a way of naming and storing a value for later use by the program, such as data from a sensor or an intermediate value used in a calculation.

#### Declaring Variables

Before they are used, all variables have to be declared. Declaring a variable means defining its type, and optionally, setting an initial value (initializing the variable). Variables do not have to be initialized (assigned a value) when they are declared, but it is often useful.

int inputVariable1;

int inputVariable2 = 0; // both are correct

Programmers should consider the size of the numbers they wish to store in choosing variable types. Variables will [roll over](http://arduino.cc/en/Reference/VariableDeclaration#VariableRollover) when the value stored exceeds the space assigned to store it. See below for an example.

#### Variable Scope

Another important choice that programmers face is where to declare variables. The specific place that variables are declared influences how various functions in a program will see the variable. This is called variable [scope](http://arduino.cc/en/Reference/Scope).

#### Initializing Variables

Variables may be initialized (assigned a starting value) when they are declared or not. It is always good programming practice however to double check that a variable has valid data in it, before it is accessed for some other purpose.

Example:

int calibrationVal = 17; // declare calibrationVal and set initial value

#### Variable Rollover

When variables are made to exceed their maximum capacity they "roll over" back to their minimum capacity, note that this happens in both directions.

int x

x = -32,768;

x = x - 1; // x now contains 32,767 - rolls over in neg. direction

x = 32,767;

x = x + 1; // x now contains -32,768 - rolls over

#### Using Variables

Once variables have been declared, they are used by setting the variable equal to the value one wishes to store with the [assignment operator](http://arduino.cc/en/Reference/Assignment) (single equal sign). The assignment operator tells the program to put whatever is on the right side of the equal sign into the variable on the left side.

inputVariable1 = 7; // sets the variable named inputVariable1 to 7

inputVariable2 = analogRead(2); // sets the variable named inputVariable2 to the

// (digitized) input voltage read from analog pin #2

#### Examples

int lightSensVal;

char currentLetter;

unsigned long speedOfLight = 186000UL;

char errorMessage = {"choose another option"}; // see string

Once a variable has been set (assigned a value), you can test its value to see if it meets certain conditions, or you can use its value directly. For instance, the following code tests whether the inputVariable2 is less than 100, then sets a delay based on inputVariable2 which is a minimum of 100:

if (inputVariable2 < 100)

{

inputVariable2 = 100;

}

delay(inputVariable2);

This example shows all three useful operations with variables. It tests the variable ( if (inputVariable2 < 100) ), it sets the variable if it passes the test ( inputVariable2 = 100 ), and it uses the value of the variable as an input parameter to the delay() function ( delay(inputVariable2) )

**Style Note:** You should give your variables descriptive names, so as to make your code more readable. Variable names like **tiltSensor** or **pushButton** help you (and anyone else reading your code) understand what the variable represents. Variable names like **var** or **value**, on the other hand, do little to make your code readable.

You can name a variable any word that is not already one of the keywords in Arduino. Avoid beginning variable names with numeral characters.

## Integer Constants

Integer constants are numbers used directly in a sketch, like 123. By default, these numbers are treated as [int](http://arduino.cc/en/Reference/Int)'s but you can change this with the U and L modifiers (see below).

Normally, integer constants are treated as base 10 (decimal) integers, but special notation (formatters) may be used to enter numbers in other bases.

Base Example Formatter Comment

10 (decimal) 123 none

2 (binary) B1111011 leading 'B' only works with 8 bit values (0 to 255)

characters 0-1 valid

8 (octal) 0173 leading "0" characters 0-7 valid

16 (hexadecimal) 0x7B leading "0x" characters 0-9, A-F, a-f valid

**Decimal** is base 10. This is the common-sense math with which you are acquainted. Constants without other prefixes are assumed to be in decimal format.

Example:

101 // same as 101 decimal ((1 \* 10^2) + (0 \* 10^1) + 1)

**Binary** is base two. Only characters 0 and 1 are valid.

Example:

B101 // same as 5 decimal ((1 \* 2^2) + (0 \* 2^1) + 1)

The binary formatter only works on bytes (8 bits) between 0 (B0) and 255 (B11111111). If it is convenient to input an int (16 bits) in binary form you can do it a two-step procedure such as:

myInt = (B11001100 \* 256) + B10101010; // B11001100 is the high byte

**Octal** is base eight. Only characters 0 through 7 are valid. Octal values are indicated by the prefix "0"

Example:

0101 // same as 65 decimal ((1 \* 8^2) + (0 \* 8^1) + 1)

Warning

It is possible to generate a hard-to-find bug by (unintentionally) including a leading zero before a constant and having the compiler unintentionally interpret your constant as octal.

**Hexadecimal (or hex)** is base sixteen. Valid characters are 0 through 9 and letters A through F; A has the value 10, B is 11, up to F, which is 15. Hex values are indicated by the prefix "0x". Note that A-F may be syted in upper or lower case (a-f).

Example:

0x101 // same as 257 decimal ((1 \* 16^2) + (0 \* 16^1) + 1)

#### U & L formatters

By default, an integer constant is treated as an [int](http://arduino.cc/en/Reference/Int) with the attendant limitations in values. To specify an integer constant with another data type, follow it with:

* a 'u' or 'U' to force the constant into an unsigned data format. Example: 33u
* a 'l' or 'L' to force the constant into a long data format. Example: 100000L
* a 'ul' or 'UL' to force the constant into an unsigned long constant. Example: 32767ul

## unsigned long

#### Description

Unsigned long variables are extended size variables for number storage, and store 32 bits (4 bytes). Unlike standard longs unsigned longs won't store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

#### Example

unsigned long time;

void setup()

{

Serial.begin(9600);

}

void loop()

{

Serial.print("Time: ");

time = millis();

//prints time since program started

Serial.println(time);

// wait a second so as not to send massive amounts of data

delay(1000);

}

#### Syntax

unsigned long var = val;

* var - your long variable name
* val - the value you assign to that variable

.

# long

#### Description

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from -2,147,483,648 to 2,147,483,647.

#### Example

long speedOfLight = 186000L; // see Integer Constants for explanation of the 'L'

#### Syntax

long var = val;

* var - the long variable name

## unsigned int

#### Description

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1).

The Due stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 (2^32 - 1).

The difference between unsigned ints and (signed) ints, lies in the way the highest bit, sometimes refered to as the "sign" bit, is interpreted. In the Arduino int type (which is signed), if the high bit is a "1", the number is interpreted as a negative number, and the other 15 bits are interpreted with [2's complement math.](http://en.wikipedia.org/wiki/2%27s_complement)

#### Example

unsigned int ledPin = 13;

#### Syntax

unsigned int var = val;

* var - your unsigned int variable name
* val - the value you assign to that variable

#### Coding Tip

When variables are made to exceed their maximum capacity they "roll over" back to their minimum capacitiy, note that this happens in both directions

unsigned int x

x = 0;

x = x - 1; // x now contains 65535 - rolls over in neg direction

x = x + 1; // x now contains 0 - rolls over

## byte

#### Description

A byte stores an 8-bit unsigned number, from 0 to 255.

#### Example

byte b = B10010; // "B" is the binary formatter (B10010 = 18 decimal)

## println()

#### Description

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as [Serial.print](http://arduino.cc/en/Serial/Print)().

#### Syntax

Serial.println(val)   
Serial.println(val, format)

#### Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

#### Returns

size\_t (long): println() returns the number of bytes written, though reading that number is optional

#### Example:

/\*  
  Analog input  
  
 reads an analog input on analog in 0, prints the value out.  
  
 created 24 March 2006  
 by Tom Igoe  
 \*/  
  
int analogValue = 0;    // variable to hold the analog value  
  
void setup() {  
  // open the serial port at 9600 bps:  
  Serial.begin(9600);  
}  
  
void loop() {  
  // read the analog input on pin 0:  
  analogValue = analogRead(0);  
  
  // print it out in many formats:  
  Serial.println(analogValue);       // print as an ASCII-encoded decimal  
  Serial.println(analogValue, DEC);  // print as an ASCII-encoded decimal  
  Serial.println(analogValue, HEX);  // print as an ASCII-encoded hexadecimal  
  Serial.println(analogValue, OCT);  // print as an ASCII-encoded octal  
  Serial.println(analogValue, BIN);  // print as an ASCII-encoded binary  
  
  // delay 10 milliseconds before the next reading:  
  delay(10);  
}

Top of Form

## if (Serial)

#### Description

Indicates if the specified Serial port is ready.

On the Leonardo, **if (Serial)** indicates wether or not the USB CDC serial connection is open. For all other instances, including **if (Serial1)** on the Leonardo, this will always returns true.

This was introduced in Arduino 1.0.1.

#### Syntax

All boards:  
if (Serial)

Arduino Leonardo specific:   
if (Serial1)   
Arduino Mega specific:   
if (Serial1)   
if (Serial2)   
if (Serial3)

#### Parameters

none

#### Returns

boolean : returns true if the specified serial port is available. This will only return false if querying the Leonardo's USB CDC serial connection before it is ready.

### Example:

void setup() {   
 //Initialize serial and wait for port to open:  
  Serial.begin(9600);   
  while (!Serial) {  
    ; // wait for serial port to connect. Needed for Leonardo only  
  }  
}   
  
void loop() {   
 //proceed normally  
}

## available()

#### Description

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes). available() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.available()

Arduino Mega only:   
Serial1.available()   
Serial2.available()   
Serial3.available()

### Parameters

none

#### Returns

the number of bytes available to read

### Example

int incomingByte = 0;   // for incoming serial data  
  
void setup() {  
        Serial.begin(9600);     // opens serial port, sets data rate to 9600 bps  
}  
  
void loop() {  
  
        // send data only when you receive data:  
        if (Serial.available() > 0) {  
                // read the incoming byte:  
                incomingByte = Serial.read();  
  
                // say what you got:  
                Serial.print("I received: ");  
                Serial.println(incomingByte, DEC);  
        }  
}

[[Get Code]](http://arduino.cc/en/Serial/Available?action=sourceblock&num=1)

**Arduino Mega example:**

void setup() {  
  Serial.begin(9600);  
  Serial1.begin(9600);  
  
}  
  
void loop() {  
  // read from port 0, send to port 1:  
  if (Serial.available()) {  
    int inByte = Serial.read();  
    Serial1.print(inByte, BYTE);   
  
  }  
  // read from port 1, send to port 0:  
  if (Serial1.available()) {  
    int inByte = Serial1.read();  
    Serial.print(inByte, BYTE);   
  }  
}

## begin()

#### Description

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.

#### Syntax

Serial.begin(speed)  
Serial.begin(speed, config)

Arduino Mega only:   
Serial1.begin(speed)   
Serial2.begin(speed)   
Serial3.begin(speed)   
Serial1.begin(speed, config)   
Serial2.begin(speed, config)   
Serial3.begin(speed, config)

#### Parameters

speed: in bits per second (baud) - long  
config: sets data, parity, and stop bits. Valid values are :

* SERIAL\_5N1
* SERIAL\_6N1
* SERIAL\_7N1
* SERIAL\_8N1 (the default)
* SERIAL\_5N2
* SERIAL\_6N2
* SERIAL\_7N2
* SERIAL\_8N2
* SERIAL\_5E1
* SERIAL\_6E1
* SERIAL\_7E1
* SERIAL\_8E1
* SERIAL\_5E2
* SERIAL\_6E2
* SERIAL\_7E2
* SERIAL\_8E2
* SERIAL\_5O1
* SERIAL\_6O1
* SERIAL\_7O1
* SERIAL\_8O1
* SERIAL\_5O2
* SERIAL\_6O2
* SERIAL\_7O2
* SERIAL\_8O2

#### Returns

nothing

### Example:

void setup() {  
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps  
}  
  
void loop() {}

[[Get Code]](http://arduino.cc/en/Serial/Begin?action=sourceblock&num=1)

**Arduino Mega example:**

// Arduino Mega using all four of its Serial ports   
// (Serial, Serial1, Serial2, Serial3),   
// with different baud rates:  
  
void setup(){  
  Serial.begin(9600);  
  Serial1.begin(38400);  
  Serial2.begin(19200);  
  Serial3.begin(4800);  
  
  Serial.println("Hello Computer");  
  Serial1.println("Hello Serial 1");  
  Serial2.println("Hello Serial 2");  
  Serial3.println("Hello Serial 3");  
}  
  
void loop() {}

## end()

#### Description

Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, call [Serial.begin](http://arduino.cc/en/Serial/Begin)().

#### Syntax

Serial.end()

Arduino Mega only:   
Serial1.end()   
Serial2.end()   
Serial3.end()

Bottom of Form

## Serial.find()

#### Description

Serial.find() reads data from the serial buffer until the target string of given length is found. The function returns true if target string is found, false if it times out.

Serial.flush() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.find(target)

#### Parameters

target : the string to search for (char)

#### Returns

boolean

## Serial.findUntil()

#### Description

Serial.findUntil() reads data from the serial buffer until a target string of given length or terminator string is found.

The function returns true if the target string is found, false if it times out.

Serial.findUntil() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.findUntil(target, terminal)

#### Parameters

target : the string to search for (char)  
terminal : the terminal string in the search (char)

#### Returns

boolean

Bottom of Form

## flush()

#### Description

Waits for the transmission of outgoing serial data to complete. (Prior to Arduino 1.0, this instead removed any buffered incoming serial data.)

flush() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.flush()

Arduino Mega only:   
Serial1.flush()   
Serial2.flush()   
Serial3.flush()

#### Parameters

none

#### Returns

nothing

## Serial.parseFloat()

#### Description

Serial.parseFloat() returns the first valid floating point number from the Serial buffer. Characters that are not digits (or the minus sign) are skipped. parseFloat() is terminated by the first character that is not a floating point number.

Serial.parseFloat() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.parseFloat()

#### Parameters

none

#### Returns

float

## parseInt()

#### Description

Looks for the next valid integer in the incoming serial stream. parseInt() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

If no valid integer is found within one second (adjustable through [Serial.setTimeout](http://arduino.cc/en/Serial/SetTimeout)() ) a default value of 0 will be returned.

#### Syntax

Serial.parseInt()

Arduino Mega only:   
Serial1.parseInt()   
Serial2.parseInt()   
Serial3.parseInt()

### Parameters

none

#### Returns

int : the next valid integer

## peek()

#### Description

Returns the next byte (character) of incoming serial data without removing it from the internal serial buffer. That is, successive calls to peek() will return the same character, as will the next call to read(). peek() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.peek()

Arduino Mega only:   
Serial1.peek()   
Serial2.peek()   
Serial3.peek()

### Parameters

None

#### Returns

the first byte of incoming serial data available (or -1 if no data is available) - int

## print()

#### Description

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example:

* Serial.print(78) gives "78"
* Serial.print(1.23456) gives "1.23"
* Serial.print('N') gives "N"
* Serial.print("Hello world.") gives "Hello world."

An optional second parameter specifies the base (format) to use; permitted values are BIN (binary, or base 2), OCT (octal, or base 8), DEC (decimal, or base 10), HEX (hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example:

* Serial.print(78, BIN) gives "1001110"
* Serial.print(78, OCT) gives "116"
* Serial.print(78, DEC) gives "78"
* Serial.print(78, HEX) gives "4E"
* Serial.println(1.23456, 0) gives "1"
* Serial.println(1.23456, 2) gives "1.23"
* Serial.println(1.23456, 4) gives "1.2346"

You can pass flash-memory based strings to Serial.print() by wrapping them with F(). For example :

* Serial.print(F(“Hello World”))

To send a single byte, use [Serial.write](http://arduino.cc/en/Serial/Write)().

#### Syntax

Serial.print(val)   
Serial.print(val, format)

#### Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

#### Returns

size\_t (long): print() returns the number of bytes written, though reading that number is optional

#### Example:

/\*  
Uses a FOR loop for data and prints a number in various formats.  
\*/  
int x = 0;    // variable  
  
void setup() {  
  Serial.begin(9600);      // open the serial port at 9600 bps:      
}  
  
void loop() {    
  // print labels   
  Serial.print("NO FORMAT");       // prints a label  
  Serial.print("\t");              // prints a tab  
  
  Serial.print("DEC");    
  Serial.print("\t");        
  
  Serial.print("HEX");   
  Serial.print("\t");     
  
  Serial.print("OCT");  
  Serial.print("\t");  
  
  Serial.print("BIN");  
  Serial.print("\t");   
  
  for(x=0; x< 64; x++){    // only part of the ASCII chart, change to suit  
  
    // print it out in many formats:  
    Serial.print(x);       // print as an ASCII-encoded decimal - same as "DEC"  
    Serial.print("\t");    // prints a tab  
  
    Serial.print(x, DEC);  // print as an ASCII-encoded decimal  
    Serial.print("\t");    // prints a tab  
  
    Serial.print(x, HEX);  // print as an ASCII-encoded hexadecimal  
    Serial.print("\t");    // prints a tab  
  
    Serial.print(x, OCT);  // print as an ASCII-encoded octal  
    Serial.print("\t");    // prints a tab  
  
    Serial.println(x, BIN);  // print as an ASCII-encoded binary  
    //                             then adds the carriage return with "println"  
    delay(200);            // delay 200 milliseconds  
  }  
  Serial.println("");      // prints another carriage return  
}

#### Programming Tips

As of version 1.0, serial transmission is asynchronous; Serial.print() will return before any characters are transmitted.

## println()

#### Description

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as [Serial.print](http://arduino.cc/en/Serial/Print)().

#### Syntax

Serial.println(val)   
Serial.println(val, format)

#### Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

#### Returns

size\_t (long): println() returns the number of bytes written, though reading that number is optional

#### Example:

/\*  
  Analog input  
  
 reads an analog input on analog in 0, prints the value out.  
  
 created 24 March 2006  
 by Tom Igoe  
 \*/  
  
int analogValue = 0;    // variable to hold the analog value  
  
void setup() {  
  // open the serial port at 9600 bps:  
  Serial.begin(9600);  
}  
  
void loop() {  
  // read the analog input on pin 0:  
  analogValue = analogRead(0);  
  
  // print it out in many formats:  
  Serial.println(analogValue);       // print as an ASCII-encoded decimal  
  Serial.println(analogValue, DEC);  // print as an ASCII-encoded decimal  
  Serial.println(analogValue, HEX);  // print as an ASCII-encoded hexadecimal  
  Serial.println(analogValue, OCT);  // print as an ASCII-encoded octal  
  Serial.println(analogValue, BIN);  // print as an ASCII-encoded binary  
  
  // delay 10 milliseconds before the next reading:  
  delay(10);  
}

Top of Form

## read()

#### Description

Reads incoming serial data. read() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.read()

Arduino Mega only:   
Serial1.read()   
Serial2.read()   
Serial3.read()

### Parameters

None

#### Returns

the first byte of incoming serial data available (or -1 if no data is available) - int

### Example

int incomingByte = 0;   // for incoming serial data  
  
void setup() {  
        Serial.begin(9600);     // opens serial port, sets data rate to 9600 bps  
}  
  
void loop() {  
  
        // send data only when you receive data:  
        if (Serial.available() > 0) {  
                // read the incoming byte:  
                incomingByte = Serial.read();  
  
                // say what you got:  
                Serial.print("I received: ");  
                Serial.println(incomingByte, DEC);  
        }  
}

## Serial.readBytes()

#### Description

Serial.readBytes() reads characters from the serial port into a buffer. The function terminates if the determined length has been read, or it times out (see [Serial.setTimeout](http://arduino.cc/en/Serial/SetTimeout)()).

Serial.readBytes() returns the number of characters placed in the buffer. A 0 means no valid data was found.

Serial.readBytes() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.readBytes(buffer, length)

#### Parameters

buffer: the buffer to store the bytes in (char[] or byte[])  
length : the number of bytes to read (int)

#### Returns

byte

## Serial.readBytesUntil()

#### Description

Serial.readBytesUntil() reads characters from the serial buffer into an array. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see [Serial.setTimeout](http://arduino.cc/en/Serial/SetTimeout)()).

Serial.readBytesUntil() returns the number of characters read into the buffer. A 0 means no valid data was found.

Serial.readBytesUntil() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.readBytesUntil(character, buffer, length)

#### Parameters

character : the character to search for (char)  
buffer: the buffer to store the bytes in (char[] or byte[]) length : the number of bytes to read (int)

#### Returns

byte

## Serial.setTimeout()

#### Description

Serial.setTimeout() sets the maximum milliseconds to wait for serial data when using [Serial.readBytesUntil](http://arduino.cc/en/Serial/ReadBytesUntil)() or [Serial.readBytes](http://arduino.cc/en/Serial/ReadBytes)(). It defaults to 1000 milliseconds.

Serial.setTimeout() inherits from the [Stream](http://arduino.cc/en/Reference/Stream) utility class.

#### Syntax

Serial.setTimeout(time)

#### Parameters

time : timeout duration in milliseconds (long).

#### Parameters

## write()

#### Description

Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the [print](http://arduino.cc/en/Serial/Print)() function instead.

#### Syntax

Serial.write(val)   
Serial.write(str)   
Serial.write(buf, len)

Arduino Mega also supports: Serial1, Serial2, Serial3 (in place of Serial)

#### Parameters

val: a value to send as a single byte

str: a string to send as a series of bytes

buf: an array to send as a series of bytes

len: the length of the buffer

#### Returns

byte  
write() will return the number of bytes written, though reading that number is optional

#### Example

void setup(){  
  Serial.begin(9600);  
}  
  
void loop(){  
  Serial.write(45); // send a byte with the value 45  
  
   int bytesSent = Serial.write(“hello”); //send the string “hello” and return the length of the string.  
}

#### 2.Digital

* [Blink Without Delay](http://arduino.cc/en/Tutorial/BlinkWithoutDelay): blinking an LED without using the delay() function.
* [Button](http://arduino.cc/en/Tutorial/Button): use a pushbutton to control an LED.
* [Debounce](http://arduino.cc/en/Tutorial/Debounce): read a pushbutton, filtering noise.
* [Button State Change](http://arduino.cc/en/Tutorial/ButtonStateChange): counting the number of button pushes.
* [Input Pullup Serial](http://arduino.cc/en/Tutorial/InputPullupSerial): Demonstrates the use of INPUT\_PULLUP with pinMode().
* [Tone](http://arduino.cc/en/Tutorial/Tone): play a melody with a Piezo speaker.
* [Pitch follower](http://arduino.cc/en/Tutorial/Tone2): play a pitch on a piezo speaker depending on an analog input.
* [Simple Keyboard](http://arduino.cc/en/Tutorial/Tone3): a three-key musical keyboard using force sensors and a piezo speaker.
* [Tone4](http://arduino.cc/en/Tutorial/Tone4): play tones on multiple speakers sequentially using the tone() command.

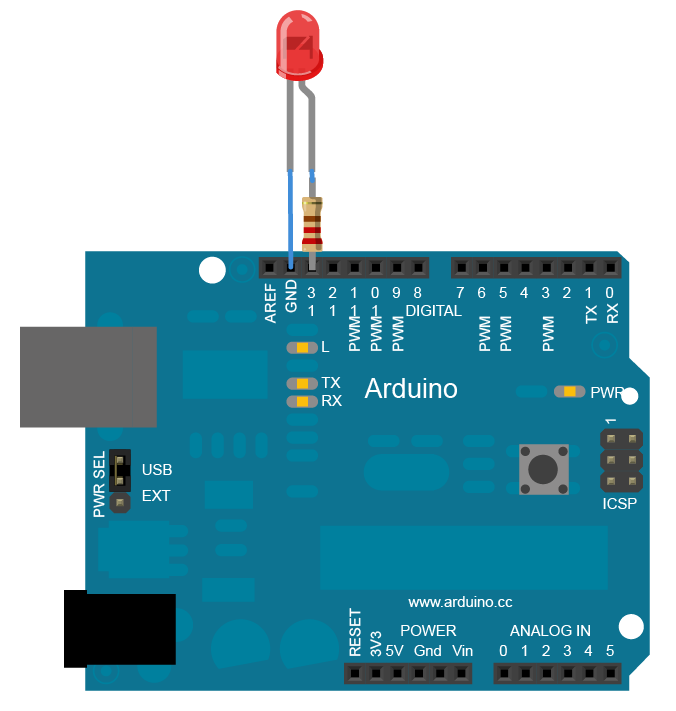
**Blink Without Delay**

Sometimes you need to do two things at once. For example you might want to blink an LED (or some other time-sensitive function) while reading a button press or other input. In this case, you can't use delay(), or you'd stop everything else the program while the LED blinked. The program might miss the button press if it happens during the delay(). This sketch demonstrates how to blink the LED without using delay(). It keeps track of the last time the Arduino turned the LED on or off. Then, each time through loop(), it checks if a long enough interval has passed. If it has, it toggles the LED on or off.

**Hardware Required**

* Arduino Board
* LED

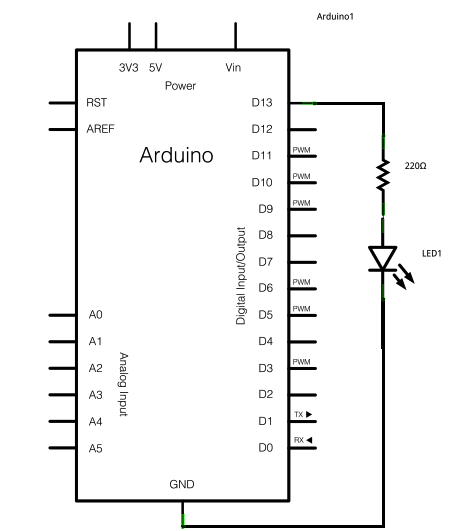
**Circuit**

[](http://arduino.cc/en/uploads/Tutorial/ExampleCircuit_bb.png)

To build the circuit, grab an LED and attach its long, positive leg (called the anode) to pin 13. Attach the short, negative leg (called the cathode) to ground. Then plug your Arduino board into your computer, start the Arduino program, and enter the code below.

**Schematic:**

click the image to enlarge

[](http://arduino.cc/en/uploads/Tutorial/ExampleCircuit_sch.png)

**Code**

The code below uses the [millis()](http://arduino.cc/en/Reference/Millis) function, a command that returns the number of milliseconds since the Arduino board started running its current program, to blink an LED.

/\* Blink without Delay  
   
 Turns on and off a light emitting diode(LED) connected to a digital    
 pin, without using the delay() function.  This means that other code  
 can run at the same time without being interrupted by the LED code.  
   
 The circuit:  
 \* LED attached from pin 13 to ground.  
 \* Note: on most Arduinos, there is already an LED on the board  
 that's attached to pin 13, so no hardware is needed for this example.  
   
   
 created 2005  
 by David A. Mellis  
 modified 8 Feb 2010  
 by Paul Stoffregen  
   
 This example code is in the public domain.  
  
   
 http://www.arduino.cc/en/Tutorial/BlinkWithoutDelay  
 \*/  
  
// constants won't change. Used here to   
// set pin numbers:  
const int ledPin =  13;      // the number of the LED pin  
  
// Variables will change:  
int ledState = LOW;             // ledState used to set the LED  
long previousMillis = 0;        // will store last time LED was updated  
  
// the follow variables is a long because the time, measured in miliseconds,  
// will quickly become a bigger number than can be stored in an int.  
long interval = 1000;           // interval at which to blink (milliseconds)  
  
void setup() {  
  // set the digital pin as output:  
  pinMode(ledPin, OUTPUT);        
}  
  
void loop()  
{  
  // here is where you'd put code that needs to be running all the time.  
  
  // check to see if it's time to blink the LED; that is, if the   
  // difference between the current time and last time you blinked   
  // the LED is bigger than the interval at which you want to   
  // blink the LED.  
  unsigned long currentMillis = millis();  
   
  if(currentMillis - previousMillis > interval) {  
    // save the last time you blinked the LED   
    previousMillis = currentMillis;     
  
    // if the LED is off turn it on and vice-versa:  
    if (ledState == LOW)  
      ledState = HIGH;  
    else  
      ledState = LOW;  
  
    // set the LED with the ledState of the variable:  
    digitalWrite(ledPin, ledState);  
  }  
}

Top of Form

**Bare Minimum code needed to get started**

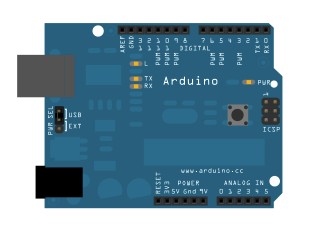
This example contains the bare minimum of code you need for an Arduino sketch to compile: the setup() method and the loop() method.

**Hardware Required**

* Arduino Board

**Circuit**

Only your Arduino Board is needed for this example.

[](http://arduino.cc/en/uploads/Tutorial/Arduino_bb.png)

**Code**

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

After creating a setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond as it runs. Code in the loop() section of your sketch is used to actively control the Arduino board.

The code below won't actually do anything, but it's structure is useful for copying and pasting to get you started on any sketch of your own. It also shows you how to make comments in your code.

Any line that starts with two slashes (//) will not be read by the compiler, so you can write anything you want after it. Commenting your code like this can be particularly helpful in explaining, both to yourself and others, how your program functions step by step.

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

Top of Form

## millis()

#### Description

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

#### Parameters

None

#### Returns

Number of milliseconds since the program started (unsigned long)

#### Example

unsigned long time;

void setup(){

Serial.begin(9600);

}

void loop(){

Serial.print("Time: ");

time = millis();

//prints time since program started

Serial.println(time);

// wait a second so as not to send massive amounts of data

delay(1000);

}

#### Tip:

Note that the parameter for millis is an unsigned long, errors may be generated if a programmer tries to do math with other datatypes such as ints.

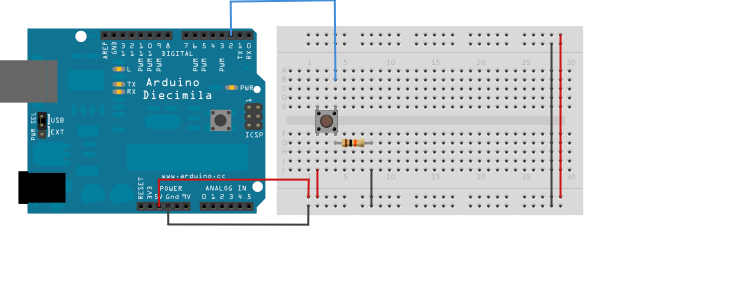
**Button**

Pushbuttons or switches connect two points in a circuit when you press them. This example turns on the built-in LED on pin 13 when you press the button.

**Hardware**

* Arduino Board
* momentary button or switch
* 10K ohm resistor
* breadboard
* hook-up wire

**Circuit**



Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 KOhms) to ground. The other leg of the button connects to the 5 volt supply.

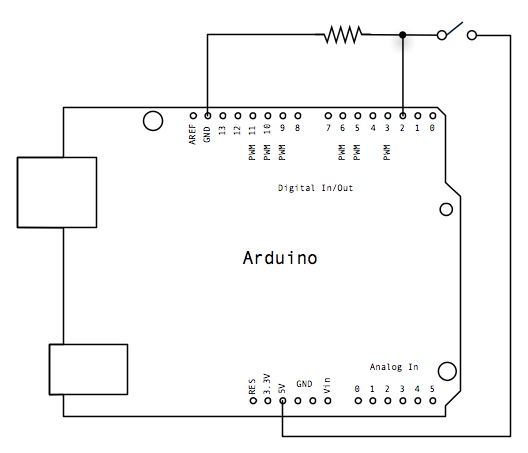
When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that we read a HIGH.

You can also wire this circuit the opposite way, with a pullup resistor keeping the input HIGH, and going LOW when the button is pressed. If so, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resistor in the circuit.

**Schematic:**

click the image to enlarge

[](http://arduino.cc/en/uploads/Tutorial/button_schem.png)

**Code**

/\*  
  Button  
   
 Turns on and off a light emitting diode(LED) connected to digital    
 pin 13, when pressing a pushbutton attached to pin 2.   
   
   
 The circuit:  
 \* LED attached from pin 13 to ground   
 \* pushbutton attached to pin 2 from +5V  
 \* 10K resistor attached to pin 2 from ground  
   
 \* Note: on most Arduinos there is already an LED on the board  
 attached to pin 13.  
   
   
 created 2005  
 by DojoDave <http://www.0j0.org>  
 modified 30 Aug 2011  
 by Tom Igoe  
   
 This example code is in the public domain.  
   
 http://www.arduino.cc/en/Tutorial/Button  
 \*/  
  
// constants won't change. They're used here to   
// set pin numbers:  
const int buttonPin = 2;     // the number of the pushbutton pin  
const int ledPin =  13;      // the number of the LED pin  
  
// variables will change:  
int buttonState = 0;         // variable for reading the pushbutton status  
  
void setup() {  
  // initialize the LED pin as an output:  
  pinMode(ledPin, OUTPUT);        
  // initialize the pushbutton pin as an input:  
  pinMode(buttonPin, INPUT);       
}  
  
void loop(){  
  // read the state of the pushbutton value:  
  buttonState = digitalRead(buttonPin);  
  
  // check if the pushbutton is pressed.  
  // if it is, the buttonState is HIGH:  
  if (buttonState == HIGH) {       
    // turn LED on:      
    digitalWrite(ledPin, HIGH);    
  }   
  else {  
    // turn LED off:  
    digitalWrite(ledPin, LOW);   
  }  
}

## pinMode()

#### Description

Configures the specified pin to behave either as an input or an output. See the description of [digital pins](http://arduino.cc/en/Tutorial/DigitalPins) for details on the functionality of the pins.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT\_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.

#### Syntax

pinMode(pin, mode)

#### Parameters

pin: the number of the pin whose mode you wish to set

mode: [INPUT](http://arduino.cc/en/Reference/Constants), [OUTPUT](http://arduino.cc/en/Reference/Constants), or [INPUT\_PULLUP](http://arduino.cc/en/Reference/Constants). (see the [digital pins](http://arduino.cc/en/Tutorial/DigitalPins) page for a more complete description of the functionality.)

#### Returns

None

#### Example

int ledPin = 13;                 // LED connected to digital pin 13  
  
void setup()  
{  
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output  
}  
  
void loop()  
{  
  digitalWrite(ledPin, HIGH);   // sets the LED on  
  delay(1000);                  // waits for a second  
  digitalWrite(ledPin, LOW);    // sets the LED off  
  delay(1000);                  // waits for a second  
}

## if / else

**if/else** allows greater control over the flow of code than the basic **if** statement, by allowing multiple tests to be grouped together. For example, an analog input could be tested and one action taken if the input was less than 500, and another action taken if the input was 500 or greater. The code would look like this:

if (pinFiveInput < 500)

{

// action A

}

else

{

// action B

}

**else** can proceed another **if** test, so that multiple, mutually exclusive tests can be run at the same time.

Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire if/else construction. If no test proves to be true, the default **else** block is executed, if one is present, and sets the default behavior.

Note that an **else if** block may be used with or without a terminating **else** block and vice versa. An unlimited number of such **else if** branches is allowed.

if (pinFiveInput < 500)

{

// do Thing A

}

else if (pinFiveInput >= 1000)

{

// do Thing B

}

else

{

// do Thing C

}

Another way to express branching, mutually exclusive tests, is with the [switch case](http://arduino.cc/en/Reference/SwitchCase) statement.

## if (conditional) and ==, !=, <, > (comparison operators)

**if**, which is used in conjunction with a comparison operator, tests whether a certain condition has been reached, such as an input being above a certain number. The format for an if test is:

if (someVariable > 50)

{

// do something here

}

The program tests to see if someVariable is greater than 50. If it is, the program takes a particular action. Put another way, if the statement in parentheses is true, the statements inside the brackets are run. If not, the program skips over the code.

The brackets may be omitted after an if statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement.

if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)

digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){

digitalWrite(LEDpin1, HIGH);

digitalWrite(LEDpin2, HIGH);

} // all are correct

The statements being evaluated inside the parentheses require the use of one or more operators:

### Comparison Operators:

**x == y** (x is equal to y)

**x != y** (x is not equal to y)

**x < y** (x is less than y)

**x > y** (x is greater than y)

**x <= y** (x is less than or equal to y)

**x >= y** (x is greater than or equal to y)

#### Warning:

Beware of accidentally using the single equal sign (e.g. if (x = 10) ). The single equal sign is the assignment operator, and sets x to 10 (puts the value 10 into the variable x). Instead use the double equal sign (e.g. if (x == 10) ), which is the comparison operator, and tests whether x is equal to 10 or not. The latter statement is only true if x equals 10, but the former statement will always be true.

This is because C evaluates the statement if (x=10) as follows: 10 is assigned to x (remember that the single equal sign is the [assignment operator](http://arduino.cc/en/Reference/Assignment)), so x now contains 10. Then the 'if' conditional evaluates 10, which always evaluates to TRUE, since any non-zero number evaluates to TRUE. Consequently, if (x = 10) will always evaluate to TRUE, which is not the desired result when using an 'if' statement. Additionally, the variable x will be set to 10, which is also not a desired action.

**if** can also be part of a branching control structure using the [if...else](http://arduino.cc/en/Reference/Else)] construction.