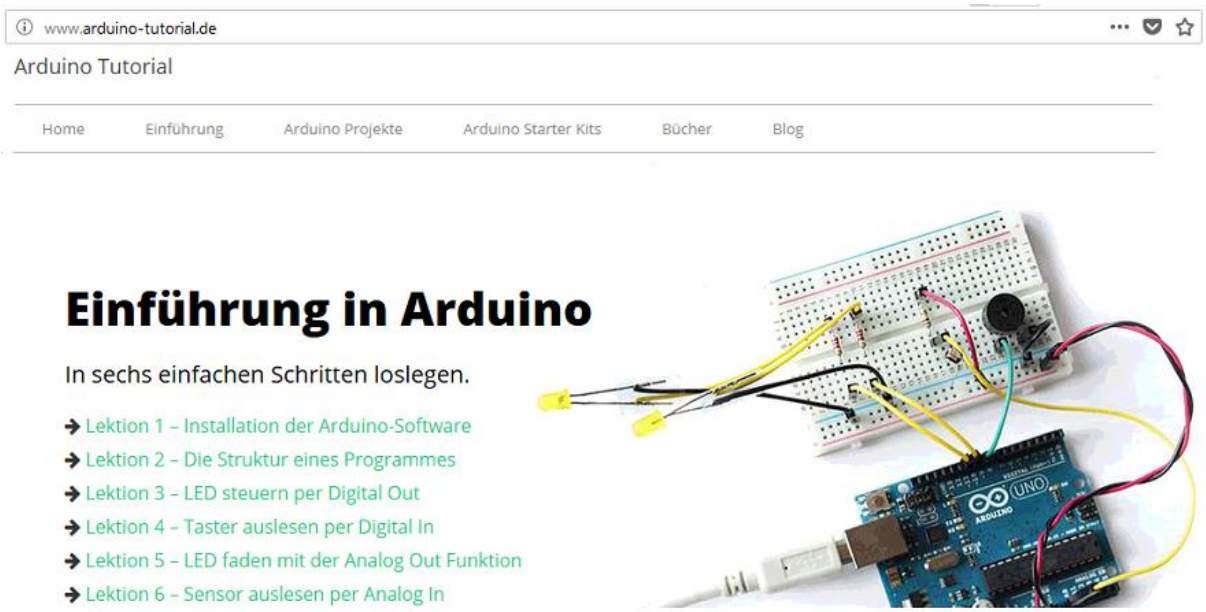
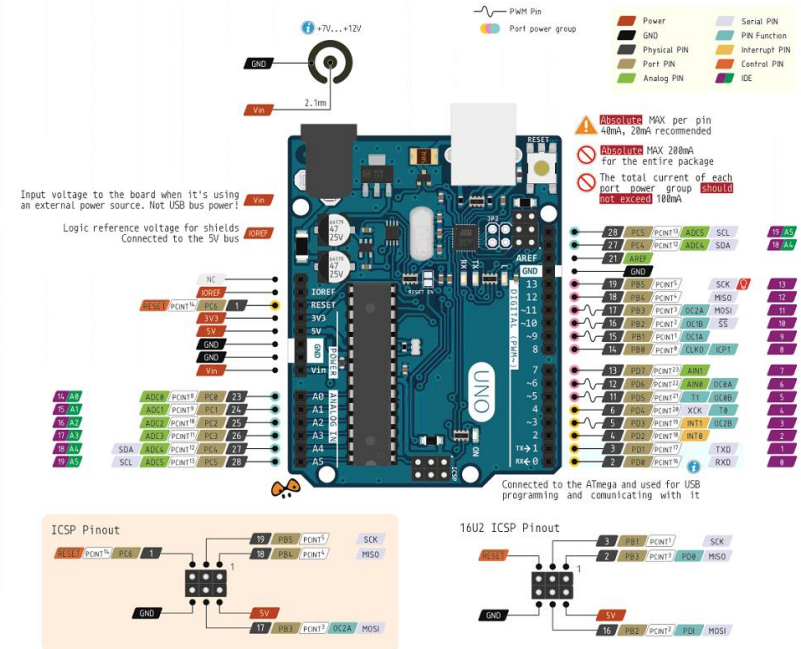
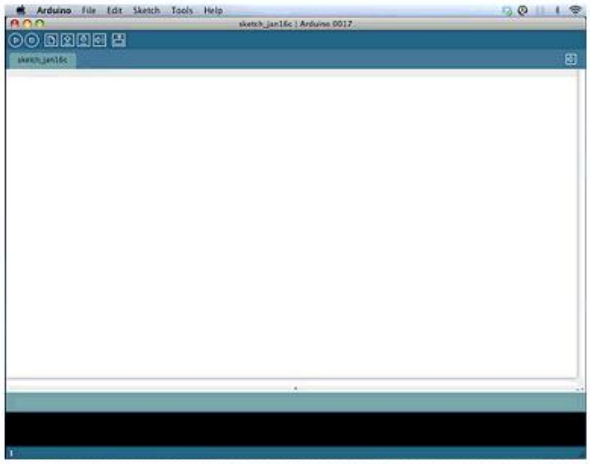
Arduino Tutorial aus dem Internet

Zum einfachen Einstieg mit dem Arduino hier eine Kurzübersicht. Details unter www.arduino-tutorial.de





Arduino Software



Screenshot der Arduinosoftware

Die Arduino Software findet man auf http://arduino.cc/en/Main/Software. Hier kann man zwischen einer Windows, Mac OS X und Linuxversion wählen. Die Software ist komplett kostenlos.

Für die Installation gibt es eine Schritt-für-Schritt-Anleitung auf der Arduino-Website.

Man benötigt zu dem eigentlichen Arduino-Programm auch einen FTDI Treiber, der mit dem Installationspaket kommt, aber separat installiert werden muss. Er dient der Datenübertragung zum Arduino-Board.

Hat man die Installation abgeschlossen, kann man das Arduino-Board an den Computer anschließen und die Arduino-Software starten.

Ganz oben befindet sich das Hauptmenü, darunter einige Symbole:

 Kompilieren (Programm auf Fehler überprüfen)

 Stop (kompilieren abbrechen)

 neuen Sketch erstellen (Programme in Arduino heißen Sketch)

 Sketch öffnen

 Sketch speichern

 Sketch auf Arduino-Board übertragen

 seriellen Monitor öffnen

Wichtig für eine korrekte Kommunikation zwischen dem Board und der Software ist, dass der richtige Port ausgewählt ist. Um den Port auszuwählen klickt man im Hauptmenü auf Tools > Serial Port und wählt dort das Arduino-Board aus der Liste aus. Bei Mac-Benutzern ist es normaler Weise der oberste Punkt in der Liste.

Nun muss noch das richtige Arduino-Modell ausgewählt werden (Tools > Board), z.B. Arduino Duemilanove.

Struktur eines Sketches

Die grundlegende Programmstruktur eines Arduino-Programms setzt sich aus zwei Methodenblöcken zusammen. Die erste Methode ist void setup(). Hier werden Grundeinstellungen (z.B. ob ein Kanal ein In- oder Output ist) vorgenommen. Diese Methode wird nur beim Programmstart ausgeführt, also genau ein Mal.

Die *void loop()* Methode wird im Gegensatz zum Setup ständig wiederholt. Hier wird der eigentliche Programmablauf geschrieben.

Über dem Setup kann man noch Bibliotheken einbinden und globale Variablen deklarieren. (Wenn du nicht weißt, was global bedeutet, mach dir noch keinen Kopf darum. Das ist für die meisten Arduino-Programme völlig egal.)

```
// Bereich für das Einbinden von Bibliotheken und die Deklaration von Variablen.

void setup() {

}

void loop() {

}
```

Weiter mit Variablen.

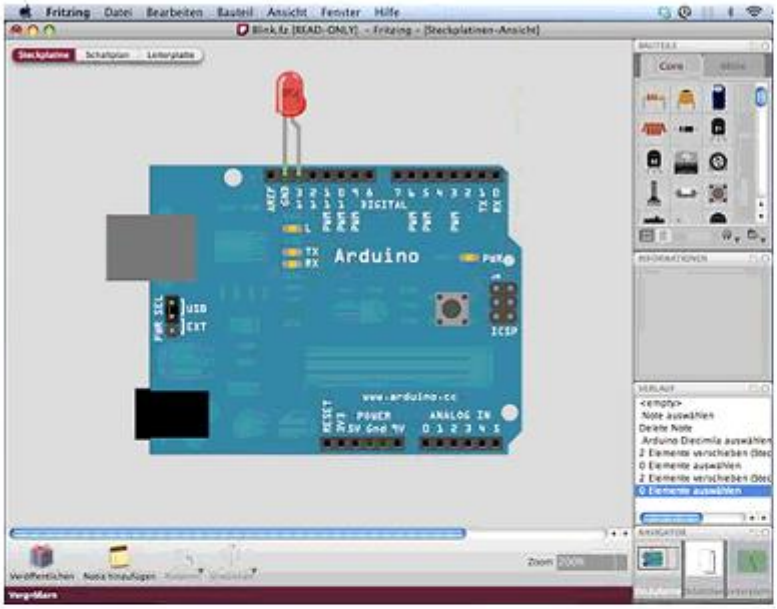
Variablen

Eine Variable ist ein Container für Werte des Typs der Variable. Variablentypen sind:

Variablentyp	Bedeutung	Beschreibung
int	ganze Zahlen	ganze Zahlen (-32.768 bis 32.767)
long	ganze Zahlen	(-2 Milliarden bis 2 Milliarden) – gut, wenn man z.B. die abgelaufenen Millisekunden zählen will, da die schon mal über 32.767 gehen
float	Fließkommazahl	gebrochene Zahlen
char	Character	Alphanumerische Zeichen (Buchstaben, Zahlen, Sonderzeichen)
array	Variablenfeld	mehrere Werte eines Variablentyps können gespeichert werden

Siehe Homepage

Digital Out



LED im GND und dem Digital 13

Digital Out *digitalOut();* ist eine Funktion, bei der ein digitaler Kanal des Arduino-Boards, das als Output deklariert ist, ein oder ausgeschaltet werden kann. Ein- oder Aus ist in diesem Fall eigentlich nicht ganz korrekt, denn der Kanal kann je nach Anweisung entweder ein 5V+ oder ein GND (Minus-Pol) sein.

Für das erste Beispiel benötigt man ein Arduino-Board und eine LED. Die LED wird mit dem kürzeren Beinchen (Kathode) in den GND-Pin (also den Minus-Pol), mit dem längeren Beinchen (Anode) in den Pin Digital 13 gesteckt.

In der Arduino-Software öffnet man das Beispiel Digital Blink (File > Examples > Digital > Blink).

Entfernt man die Kommentare, also alles, was in /* */ geschrieben ist, bleibt:

```
// by David Cuartielles

int ledPin = 13;

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

void loop()
{
digitalWrite(ledPin, HIGH);
delay(1000);
digitalWrite(ledPin, LOW);
delay(1000);
}
```

In der ersten Zeile wird eine ganzzahlige Variable (int) mit dem Namen ledPin angelegt und ihr wird der Wert 13 zugewiesen. Pin bedeutet in diesem Fall Anschluss oder Kanal. Der Befehl pinMode(ledPin, OUTPUT) setzt den digitalen Kanal 13 auf dem Arduino-Board als Output.

```
1 pinMode(13, OUTPUT);
```

Er setzt sich aus dem Methoden-Aufruf pinMode(); und der Übergabe von zwei Parametern zusammen. Parameter 1 ist die Nummer des digitalen Kanals – in diesem Fall ledPin, Parameter 2 seine Funktion OUTPUT.

In der loop-Methode stehen vier Befehle:

```
1 digitalWrite(ledPin, HIGH);
```

Dieser Befehl schaltet den ledPin auf 5V+ (HIGH);

```
1 delay(1000);
```

Dieser Befehl hält das Programm für 1000 Millisekunden an (also 1 Sekunde).

```
1 digitalWrite(ledPin, LOW);
```

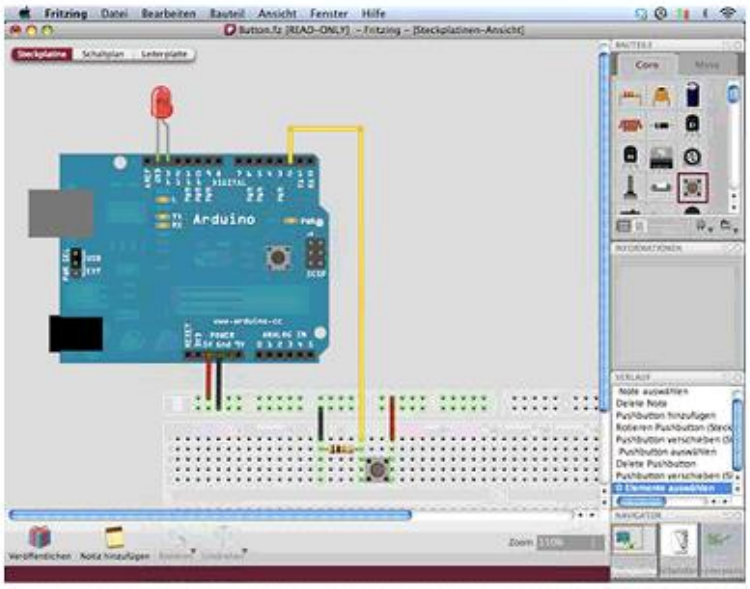
Dieser Befehl schaltet den ledPin auf GND (LOW);

```
1 delay(1000);
```

Dieser Befehl hält das Programm noch einmal für 1000 Millisekunden an.

Um das Programm auf das Arduino-Board zu laden klickt man auf das Upload Symbol und der Upload beginnt. Nun sollte die LED am Pin 13 im Takt von 2 Sekunden blinken.

Digital In



Button am Digital 2

Um ein digitales Signal zu erfassen (Schalter, Taster, usw.), erweitert man die Schaltung um einen Taster und einen Widerstand. Die eine Seite des Tasters wird mit dem 5V+ des Arduino-Boards, die andere Seite mit dem digitalen Pin 2 verbunden. Der Widerstand (1 – 10 kΩ) wird benötigt, um Spannungsschwankungen und Störsignale herauszufiltern. Man bezeichnet so einen Widerstand als Drop-Down-Widerstand. Er wird mit dem Digitalen Pin 2 und dem GND verbunden. Alle störenden Ströme werden somit über den Widerstand in den GND geführt. Ist der Taster nicht gedrückt, liegt am Pin 2 jetzt ein GND-Signal an (LOW), drückt man den Taster, so liegt ein 5V+ Signal an (HIGH).

Das Button-Beispiel aus der Arduino-Software (File > Examples > Digital > Button) sieht ohne Kommentare so aus:

```
// by DojoDave and Tom Igoe

const int buttonPin = 2;
const int ledPin =  13;
int buttonState = 0;

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

void loop(){
buttonState = digitalRead(buttonPin);
if (buttonState == HIGH) {
digitalWrite(ledPin, HIGH);
}
else {
digitalWrite(ledPin, LOW);
}
}
```

Am Anfang werden drei ganzzahlige Variablen buttonPin, ledPin und buttonState erzeugt. Dabei sind die ersten beiden unveränderlich (Schlüsselwort const für Konstante). Die Variable buttonState wird verwendet, um den aktuellen Zustand des Tasters zu speichern.

```
const int buttonPin = 2;
const int ledPin =  13;
int buttonState = 0;
```

Im Setup bekommt der ledPin die Funktion eines Outputs, der buttonPin die eines Inputs.

```
pinMode(ledPin, OUTPUT);
pinMode(buttonPin, INPUT);
```

Im Loop wird der Variable buttonState mit dem Befehl = digitalRead(buttonPin) der aktuelle Zustand des Buttons übergeben (entweder HIGH oder LOW).

```
buttonState = digitalRead(buttonPin)
```

Durch eine if-Abfrage (Alle Anweisungen innerhalb der if-Abfrage werden nur ausgeführt, wenn die in Klammern angegebene Bedingung zutrifft.) wird die LED am ledPin eingeschaltet, also wenn buttonState HIGH ist, sonst (else) wird sie abgeschaltet.

```
if (buttonState == HIGH) {
   digitalWrite(ledPin, HIGH);
 } else {
   digitalWrite(ledPin, LOW);
 }
}
```

Analog Out

Sechs der digitalen Kanäle auf dem Arduino-Boards sind nicht nur digital, sondern auch analog ansteuerbar. Sie sind mit dem Aufdruck PWM gekennzeichnet (Kanal 3, 5, 6, 9, 10, 11).

PWM (Pulse Width Modulation) bedeutet, dass kein konstantes Signal an dem Kanal anliegt, sondern dass dieser Kanal kontinuierlich an- und abgeschalten wird. In der Arduino-Software übergibt man einen Wert zwischen 0 und 255 an den Kanal. 0 entspricht dem GND (Minus-Pol), 255 entspricht 5V+, Zwischenwerte bedeuten für träge Bauteile (LEDs, Motoren, etc.) also eine Spannung zwischen 0 und 5V. Damit kann man also die Geschwindigkeit von Motoren oder die Helligkeit einer LED regulieren.

(Motoren sollten nicht direkt an einen Arduino-Kanal angeschlossen werden. Mehr dazu im Abschnitt Motorsteuerung.)

Das Arduino-Beispiel Fading (File>Examples>Analog>Fading) zeigt, wie man eine LED dimmt. Hier der Code von David A. Mellis:

```
1  // by David A. Mellis and Tom Igoe
2
3  int ledPin = 9;
4
5  void setup()  {
6
7  }
8
9  void loop()  {
10   for(int fadeValue = 0 ; fadeValue &lt;= 255; fadeValue +=5) {
11     analogWrite(ledPin, fadeValue);
12     delay(30);
13   }
14   for(int fadeValue = 255 ; fadeValue &gt;= 0; fadeValue -=5) {
15     analogWrite(ledPin, fadeValue);
16     delay(30);
17   }
18 }
```

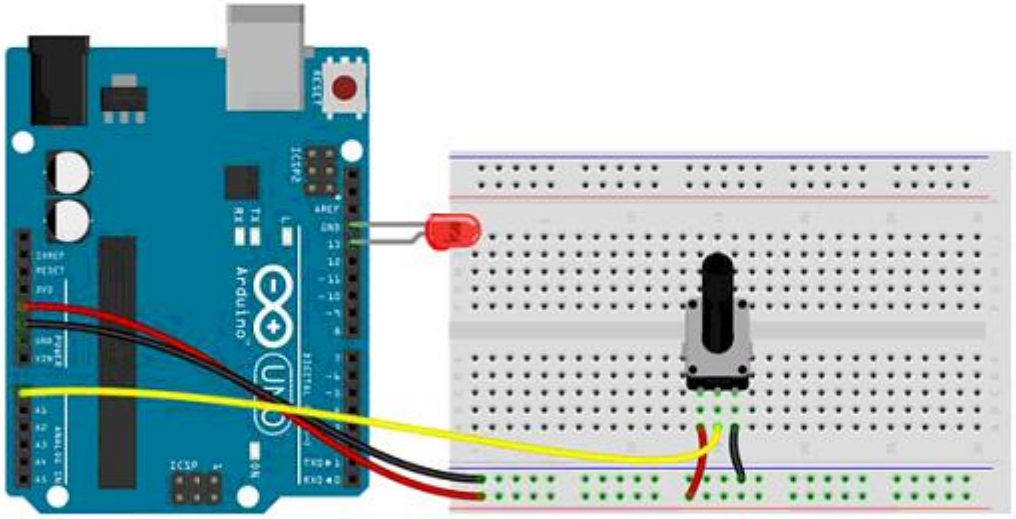
Im Gegensatz zur digitalen Ausgabe, muss ein analog angesprochener Kanal nicht im Setup deklariert werden. Der Befehl analogWrite erhält als Parameter den Pin, an dem die LED angeschlossen ist und den Wert (0 – 255), wie hell die LED leuchten soll.

Analog Input

Der Analog Input vom Arduino wird verwendet, um analoge Sensoren auszulesen. Dabei handelt es sich zum Beispiel um Potentiometer, Fotowiderstände (LDR), Druck- und Temperatursensoren. Im Gegensatz zu digitalen Signalen, die entweder HIGH oder LOW sind, liefern analoge Sensoren auch Zwischenwerte.

Analog Input Schaltplan

Im Beispiel ist ein Potentiometer ans Arduino-Board angeschlossen.



Ein Potentiometer ist am Analog Input 0 des Arduinos angeschlossen (Grafik mit Fritzing erstellt)

Die beiden äußeren Beine werden mit dem GND und dem 5V+ verbunden, das mittlere mit einem Analog Input. Das Arduino-Board kann nun das Verhältnis der Widerstände zu einander ermitteln und liefert durch den Befehl analogRead(Pin); Werte zwischen 0 und 1023.

Darüber hinaus ist eine LED mit der Anode (langes Beinchen) am Pin 13 und der Kathode (kurzes Beinchen) am nebenliegenden GND angeschlossen.

Codebeispiel

Das Beispiel AnalogInput (File>Examples>Analog>AnalogInput) aus der Arduino-Software lässt die LED verschieden schnell blinken.

```
// von David Cuartielles und Tom Igoe

int sensorPin = 0;
int ledPin = 13;
int sensorValue = 0;

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

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

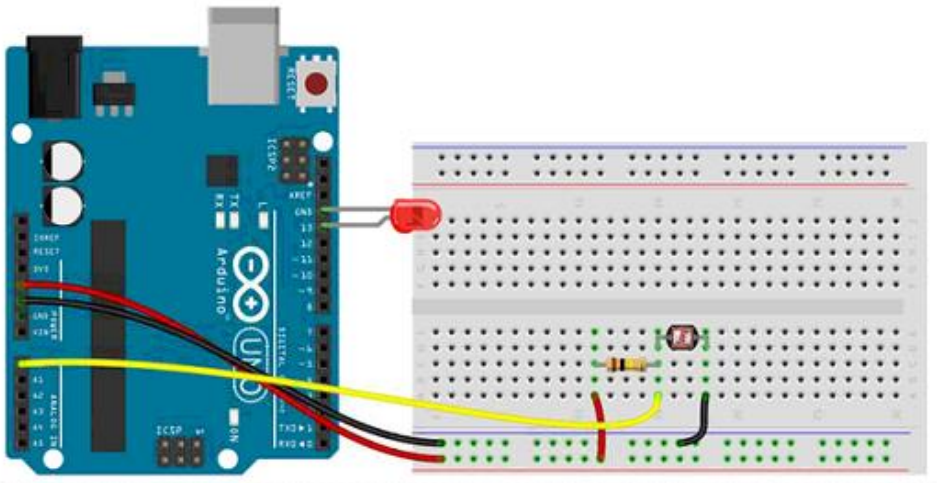
Im Beispiel wird das mittlere Bein des Potentiometers im AnalogIn 0 ausgelesen. Der Wert, der zwischen 0 und 1023 liegt, wird als Verzögerung (delay) in das Programm eingefügt und reguliert so die Blinkgeschwindigkeit der LED.

Schaltplan mit Fotowiderstand (LDR)

Nun kann man das Potentiometer auch gegen einen anderen Sensor austauschen. Wie beim Potentiometer benötigt das Arduino-Board ein Verhältnis zweier Widerstände, um einen analogen Wert zu erfassen.

Schaltplan mit Fotowiderstand (LDR)

Nun kann man das Potentiometer auch gegen einen anderen Sensor austauschen. Wie beim Potentiometer benötigt das Arduino-Board ein Verhältnis zweier Widerstände, um einen analogen Wert zu erfassen.



Fotowiderstand (LDR) am Analog Input des Arduinos (Grafik mit Fritzing erstellt.)

Ein Fotowiderstand (LDR) im Beispiel allein kann dieses Verhältnis nicht liefern. Man benötigt einen zusätzlichen Referenzwiderstand. Die Größe (Widerstandswert) des Referenzwiderstands richtet sich nach dem verwendeten Sensor und dem Umfeld, in dem er betrieben wird.

Um den Referenzwiderstand des Fotowiderstands auszurechen, muss sein Widerstand in einem hellen und einem dunklen Umfeld bestimmt werden. Beide Werte werden mit einander multipliziert und aus dem Ergebnis die Wurzel gezogen.

Wurzel aus (Rmin * Rmax) = Rref

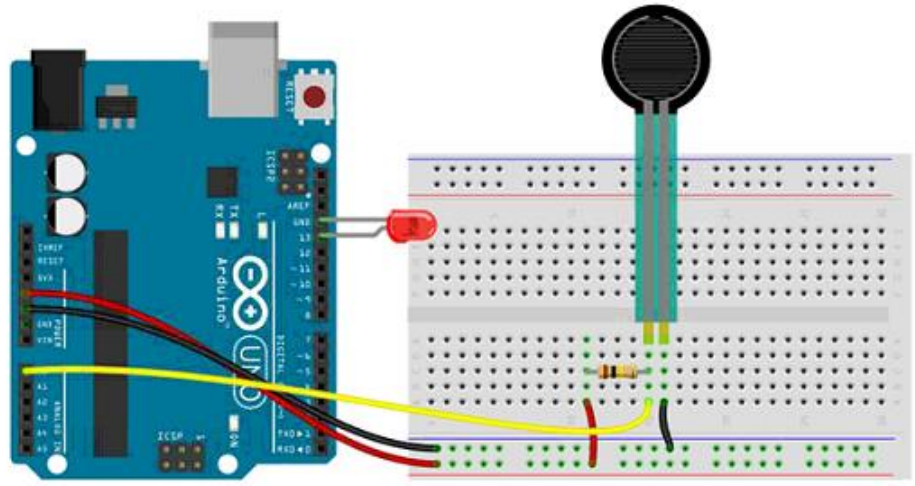
Es ergibt sich der Referenzwiderstand. Widerstände haben allerdings genormte Werte. Es reicht, einen Widerstand zu wählen, der nah dem Ausgerechneten liegt. Im Beispiel beträgt der Widerstand 100kOhm.

Alternativ kann man aber auch einfach ausprobieren, mit welchem Widerstand man ausreichende Ergebnisse erzielt.

Beispiele:

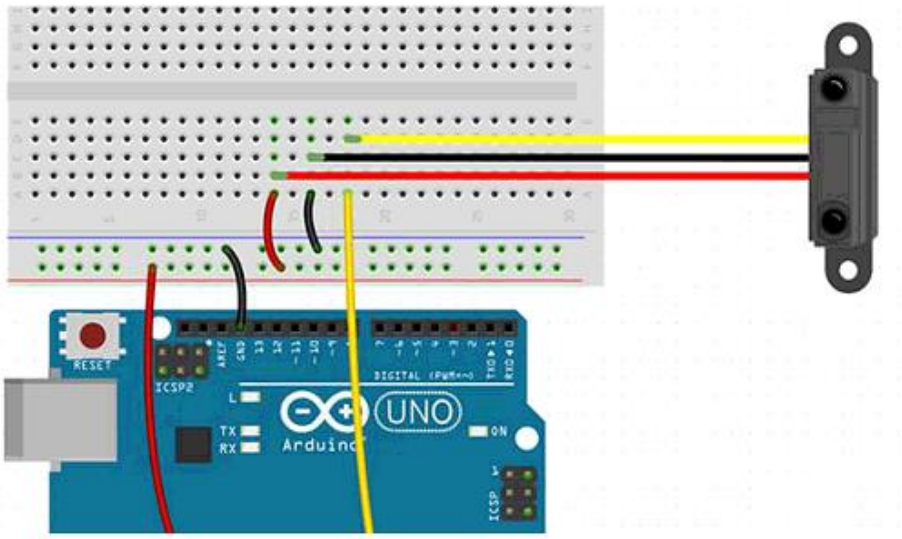
Schaltplan mit drucksensitivem Sensor (FSR)

Dieses Beispiel zeigt, wie ein drucksensitiver Sensor (FSR) angeschlossen wird. Auch für diesen Sensor wird ein Referenzwiderstand benötigt.



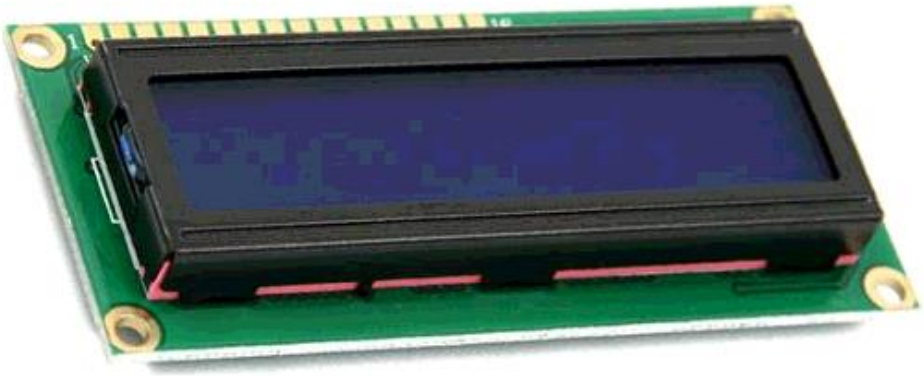
Schaltung mit drucksensitivem Sensor (Grafik mit Fritzing erstellt)

Abstandsmessung mit dem Sharp GP2Y0A21YK IR Entfernungssensor



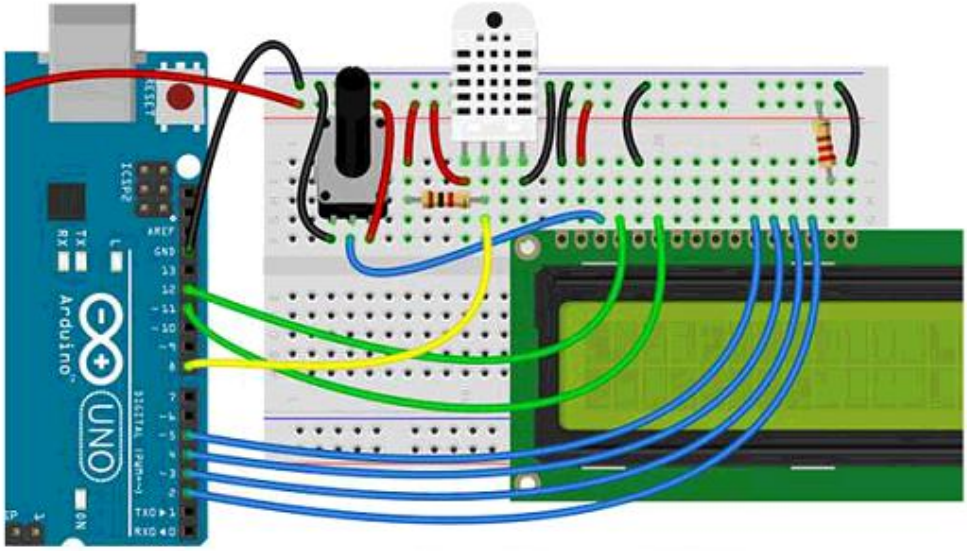
Der Infrared Proximity Sensor – Sharp GP2Y0A21YK ist ein Infrarot-Abstandssensor. Er besteht aus einer Infrarot-LED und einem Phototransistor, der das von einem Objekt reflektierte Infrarot-Licht misst. Der Sensor selbst verfügt über ...

LCD – Liquid Crystal Displays



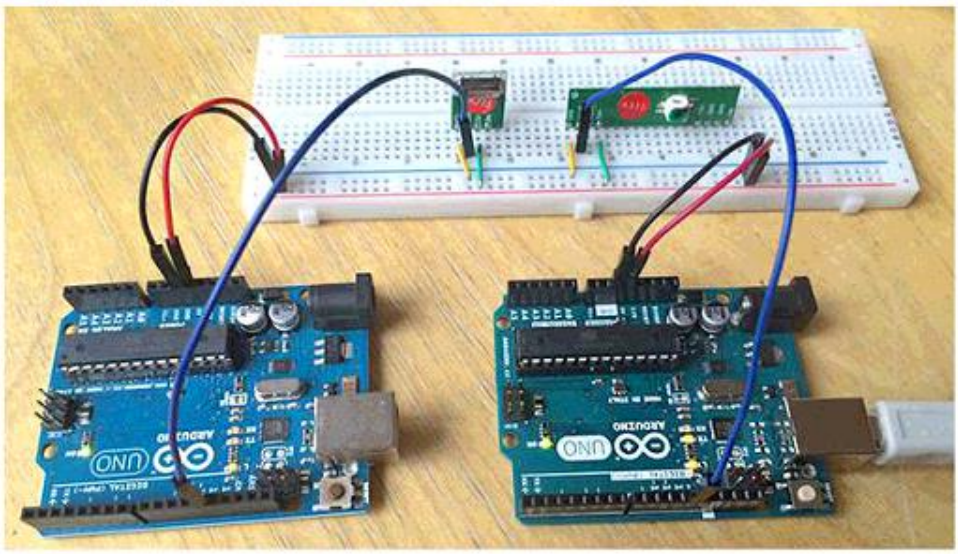
Liquid Crystal Displays (LCDs) können zur Textausgabe in Arduino-Projekten verwendet werden. Sie basieren oft auf dem HD44780 Chip von Hitachi und werden mit der Arduino-Library LiquidCrystal angesprochen. Solch ein LCD-Display* kann im 4 ...

DHT22: Temperatur und Luftfeuchtigkeit messen



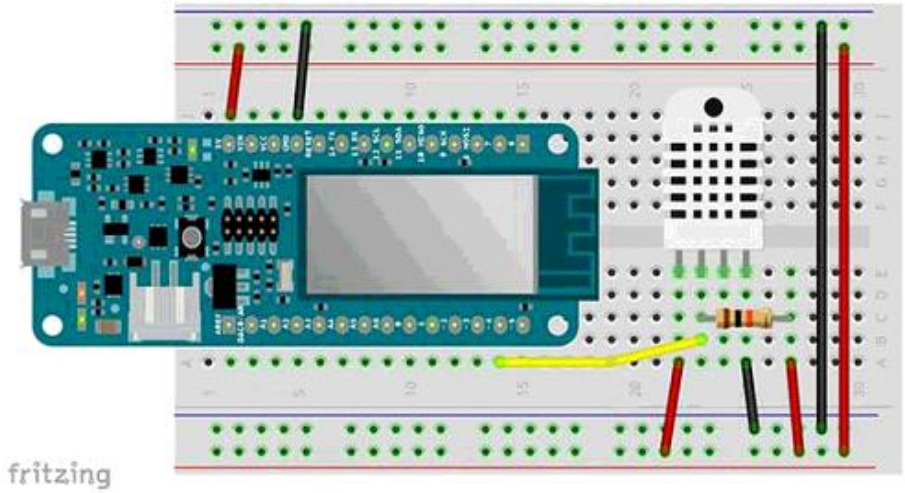
Der DHT22 (Amazon Produktlink) ist eine einfache und kostengünstige Möglichkeit, um Temperatur und Luftfeuchtigkeit zu messen. Es handelt sich um einen kalibrierten Sensor, der über eine One-Wire Schnittstelle (MaxDetect), also eine ...

Funkübertragung von Messwerten mit dem RF Link Modul



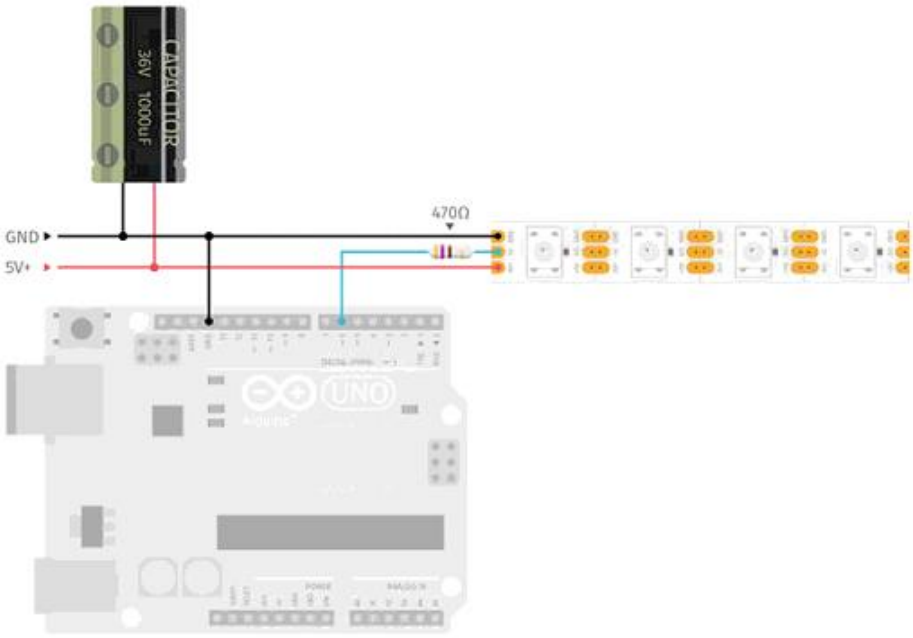
Das RF Link Modul (Receiver und Transmitter) aus der RWS-371-Serie ist ein ziemlich günstiger Weg, Signale kabellos über kurze Distanzen zu übertragen. Dieses Beispiel zeigt, wie man es verwendet. ...

Arduino zu MySQL via Wifi



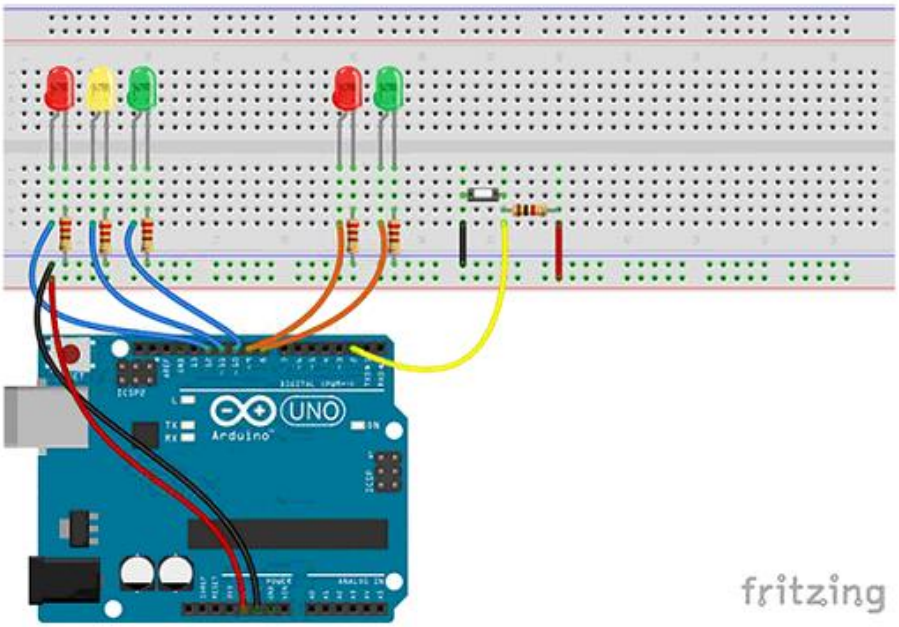
In diesem Beispiel zeige ich, wie sich Daten vom Arduino per Wifi (Wlan) in einer MySQL Datenbank speichern lassen und per Browser anzeigen lassen.
Download: Dateien auf GitHub Ich verwende ich einen ...

WS2812 – Der einfachste Weg, viele LEDs mit Arduino steuern



Bei den WS2812 LEDs handelt es sich um adressierbare RGB-LEDs. Sie verfügen über einen integrierten Chip und belegen daher nur einen einzigen digitalen Output des Arduino-Boards. Wenn man LEDs mit Arduino ...

Ampel mit Arduino für Autos und Fußgänger



Eine Ampel mit Arduino bauen ist ganz einfach. Es gibt vier Ampelphasen: Grün – Gelb – Rot – Rot-Gelb. Ampel Schaltung Code int ampelRotPin = 12; int ampelGelbPin = 11; int ampelGruenPin = 10; void setup() ...

Weitere Beispiele:

Arduino Projekt 8 – Autonomer Rennwagen

Diese Anleitung beschreibt, wie man ein RC-Spielzeugauto in ein autonom fahrendes Fahrzeug umwandelt.
Autonomer Rennwagen

Arduino Projekte mit Sensoren

Arduino Projekt 9 – Eine Einparkhilfe mit Arduino

In diesem Arduino Projekt wird gezeigt, wie man mit einem Ultraschallsensor eine Einparkhilfe mit LED-Anzeige bauen kann. Stationär in der Garage könnte das den einen oder anderen teuren Kratzer verhindern.
Eine Einparkhilfe mit Arduino

Arduino Projekt 10 – Chili Pflanzen mit dem Arduino automatisch wachsen lassen

In diesem Arduino Projekt werden Chili-Pflanzen via Arduino automatisch gepflegt. Der Code ist noch nicht veröffentlicht, aber das Projekt ist schon jetzt sehr inspirierend.
Chili Pflanzen mit dem Arduino automatisch wachsen lassen

Arduino Projekt 11 – Wetterstation mit Arduino

In diesem Arduino Projekt wird gezeigt, wie man eine Wetter-Überwachungsstation mit Arduino umsetzen kann.
Wetterstation mit Arduino

Arduino Projekt 12 – Geigerzähler mit Arduino

Nach dem Reaktorunglück in Fukushima stieg die Nachfrage nach Geigerzählern rapide. Dieses Tutorial zeigt, wie man einen Geigerzähler mit Arduino bauen kann. Ein Bausatz ist dazu allerdings zwingend erforderlich.
Geigerzähler mit Arduino

So. Ich hoffe, die Arduino Projekte gefallen euch. Viel Spaß beim Nachbauen!