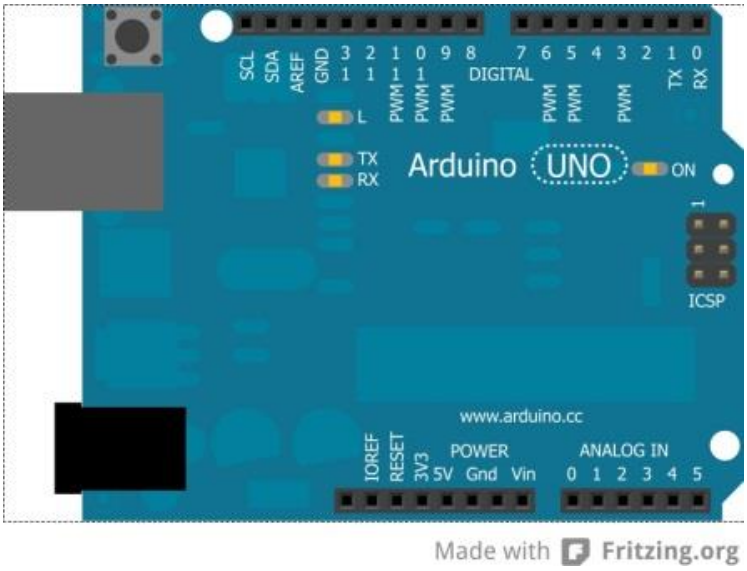
Physical Computing



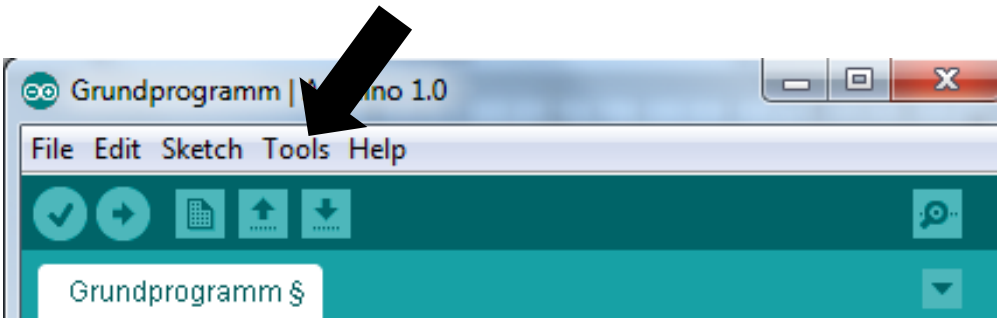
Der Arduino als Steuer-zentrale

Frank Trittler / Martin Merkle
FSG Marbach
1. Auflage
2013

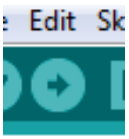
Inhalt

1. Checkliste „Erste Schritte“

- ☐ Arduino-Board mit Computer verbinden.
- ☐ Installation sicherstellen. -Notfalls im Gerätemanager Treiber nachinstallieren.
- ☐ Arduino-Programm öffnen.



- ☐ Unter ***Tools / Board*** den Arduino UNO wählen.
- ☐ Unter ***Tools / Serial Port*** den richtigen Anschluss wählen (notfalls im Gerätemanager nachschauen.
- ☐ Grundprogramm öffnen.
- ☐ Programmbeschreibung im Kopfkommentar eintragen.
- ☐ Unter einem treffenden Namen speichern.
- ☐ Programm auf den Arduino übertragen



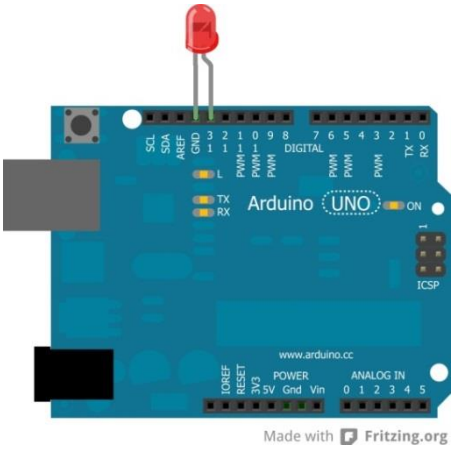
2. Grundstruktur von Programmen

Aufgabe 2.1: **Das hier beschriebene Programm hast du bereits ausprobiert. Schau dir das Programm genau an, und beschreibe an den vorgesehenen Kommentarstellen, was der Programmcode bedeutet (benutze deine guten Englisch-Kenntnisse‼).**

```
/*
Blinkprogramm.
Die eingebaute gelbe LED an Pin 13 soll blinken.
30.01.2013
 */

int LED = 13;                  //

void setup()                   //
{
  pinMode(LED, OUTPUT);        //
}

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

Aufgabe 2.2: **Schließe eine rote LED an den entsprechenden PIN an (Polung beachten) und bringe diese zum Blinken.**

Aufgabe 2.3: **Nimm noch eine grüne LED. Schließe die grüne LED an das Board an und verändere das Programm so, dass die beiden LEDs abwechselnd blinken.**



Aufgabe 2.4: **Schaffst du auch noch zwei weitere LEDs? (Tipp: die PINs können ja HIGH oder LOW geschaltet werden).**

3. Programmabschnitte

Grundprogramm §

```
/*
Blinkprogramm.
Die eingebaute gelbe LED an Pin 13 soll blinken.
30.01.2013
 */
```

```
int LED = 13;
```

```
void setup()
{
  pinMode(LED, OUTPUT);
}
```

```
void loop()
{

  digitalWrite(LED, HIGH);

  delay(1000);

  digitalWrite(LED, LOW);

  delay(1000);
  }
```

Aufgabe 3.1: **Benenne die einzelnen Programmabschnitte und beschreibe stichwortartig wozu diese jeweils dienen.**

4. Umgang mit Variablen

Variablen können ein Programm stark vereinfachen und tragen dazu bei, die Übersicht zu behalten. Verschiedenen Anschlüssen können über Variablen konkrete Namen zugeordnet werden, sodass im späteren Programm sehr einfach mit diesen Namen gearbeitet werden kann.

Tipp: Es dürfen keine Umlaute verwendet werden

Beispiel eines Programmes mit 3 LEDs:

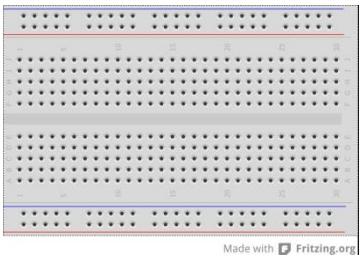
Ohne Variablen	Mit Variablen

/*
Abwechselndes Blinken
Rote LED an PIN 13, Grüne LED an PIN 12
30.01.2013
*/
 	
void setup() {
 	
pinMode(13, OUTPUT);
pinMode(12, OUTPUT);
}
 	
void loop() {
digitalWrite(13, HIGH);
digitalWrite(12, LOW);
delay(1000);
digitalWrite(13, LOW);
digitalWrite(12, HIGH);
delay(1000);
 	
}
	/*
	Abwechselndes Blinken
	Rote LED an PIN 13, Grüne LED an PIN 12
	30.01.2013
	*/
	
	int ledgruen = 13;
	int ledrot = 12;
	
	void setup() {
	
	pinMode(ledgruen, OUTPUT);
	pinMode(ledrot, OUTPUT);
	}
	
	void loop() {
	digitalWrite(ledgruen, HIGH);
	digitalWrite(ledrot, LOW);
	delay(1000);
	digitalWrite(ledgruen, LOW);
	digitalWrite(ledrot, HIGH);
	delay(1000);
	
	}

Aufgabe 4.1: Kannst du eine Variable definieren, mit deren Hilfe du die Blinkdauer einfacher einstellen kannst?

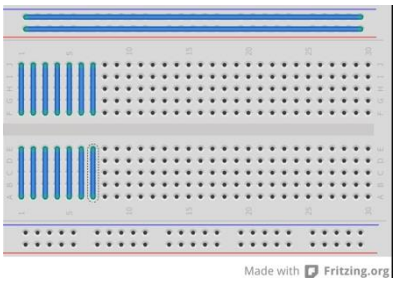
Aufgabe 4.2: Programmiere eine Ampelschaltung mit 3 LEDs.

5. Verwendung der Steckplatine - Anschluss mehrerer Bauteile



Wenn man mehrere verschiedene Bauteile an den Arduino anschließen will, so verwendet man eine Steckplatine (auch Breadboard oder Protoboard).

Auf dieser Steckplatine ist genügend Platz, um elektrische Schaltungen aufzubauen. Die Bauteile können dabei einfach in die Steckplatine hineingesteckt werden.

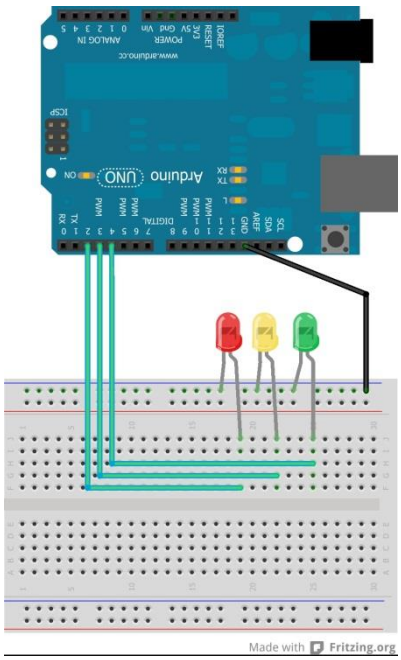


Das Besondere an der Steckplatine ist, dass die Steckplätze untereinander auf eine bestimmte Weise leitend miteinander verbunden sind. Die Verbindung ist hier links dargestellt.

So können die einzelnen Bauteile sehr einfach miteinander verbunden werden.

Zum Anschluss an den Arduino nutzt man entweder einfachen Anschlussdraht, den man an den Enden ein kurzes Stück abisoliert, oder man verwendet fertig konfektionierte Anschlusslitze mit Pins.

Als Beispiel ist hier eine Ampelschaltung dargestellt:

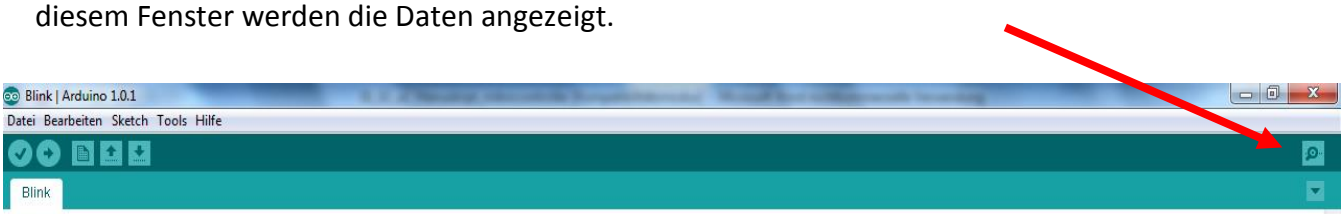


Um eine bessere Übersicht zu behalten, benutzt man ganz bestimmte ***Drahtfarben***:

schwarz	GND 0V
rot	+5V Dauerplus
grün	Ausgangssignal
gelb	Eingangssignal
blau	Leistungsausgang für Aktoren
weiss	Leistungseingang für Aktoren
orange	Pluspol der Fremdspannungsquelle

6. Datenausgabegeräte – der serielle Monitor

Die Verwendung des seriellen Monitors ist besonders praktisch, wenn man die eingehenden Daten, die von dem Board gesammelt oder produziert werden, anzeigen möchte. Um die Daten auf dem Monitor sehen zu können, muss man das Icon für den seriellen Monitor (Serial Monitor) anklicken, dann öffnet sich ein neues Fenster. In diesem Fenster werden die Daten angezeigt.



Natürlich ist es auch möglich einen Text auszugeben.

Beispiel: Hallo Welt!

```
/* Hallo Welt!
  dieser Text soll auf dem Bildschirm des PCs angezeigt werden
  Nick Teck
  1.06.2012
*/


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

  Serial.println("Hallo Welt!");                //
}

void loop()
{
                                //
}
```

Übersicht

Bibliotheksname	•	Befehl	(Eingabewert)	
Serial	.	begin	(9600)	
Serial	.	println	("Text“)	
Serial	.	print	("Text“)	

Die Anführungszeichen in der Klammer sind Teil des Befehls, wenn ein Text ausgegeben werden soll. Für die Ausgabe einer Variablen sind sie nicht notwendig.

Aufgabe 6.1: Kommentiere in der freien Spalte, was der jeweilige Befehl macht.

Aufgabe 6.2: Lasst einen beliebigen Text einmal auf dem Bildschirm anzeigen.

Aufgabe 6.3: Derselbe Text soll nun jede Sekunde noch einmal dazukommen.

7. Der Arduino kann rechnen

Der Arduino kann mit Variablen rechnen. Beispielsweise kann die Blinkdauer aus der Addition zweier Variablen errechnet werden:

delay (a+b);

Aufgabe 7.1: Schreibe ein Programm, mit dem man das Ergebnis einer Addition am Bildschirm anzeigen kann.

Aufgabe 7.2: Führe noch weitere Rechenoperationen durch.

8. Zählschleifen

Ein Programm besteht aus vier Abschnitten. Diese vier Abschnitte hast du als Grundstruktur eines Programms auswendig gelernt. Der letzte Abschnitt war die Endlosschleife [void loop()]. In diesem Programmabschnitt werden die eingetragenen Befehle endlos wiederholt. Manchmal möchte man aber Befehle nur eine bestimmte Anzahl oft wiederholen (z.B. eine rote LED soll nur genau fünfmal blinken!) Wie oft ein Befehl wiederholt werden soll, kann man mit einer sogenannten **Zähl- oder for-Schleife** festlegen.

A5_ForSchleife_Zaehlen §

```
/*
 For-Schleife

 Zählen
 Nick Tech
 20.02.2013

 */

int Zeit = 1000;           // Zähltakt
int Anfang = 0;            // Anfang wird auf den Wert Null gesetzt


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

void loop()
  {

    for (Anfang; Anfang<11; Anfang++)
  {
    Serial.println(Anfang);
    delay(Zeit);
  }

    Serial.println("Fertig");
  }
```

Aufgabe 8.1: **Kommentiere die dir bekannten Befehle im Hauptprogramm [void loop()]!**

Aufgabe 8.2: **Teste das Programm! Bestimme wie oft die for-Schleife durchlaufen wird!**

Aufgabe 8.3: **Die for-Schleife soll nun 25mal durchlaufen werden. Ändere das Programm entsprechend.**

Aufgabe 8.4: **Was bedeutet die Anweisung Anfang++?**

Aufgabe 8.5: Was bedeutet die Anweisung Anfang--?

Aufgabe 8.6: Füge die Begriffe Veränderung, Laufbedingung und Startwert, entsprechend ihren Bedeutungen, an der richtigen Stelle ein.

```
for (                    ;                    ;                    )
```

Übungen:

1. **Lauflicht**
   8 LEDs sollen nacheinander aufleuchten. Nachdem die letzte LED geleuchtet hat, soll das Aufleuchten bei der ersten LED erneut starten.

2. **Lauflicht mit Umkehr**
   Folgende Änderung soll an deinem Programm vorgenommen werden. Nachdem die letzte LED geleuchtet hat, soll das Aufleuchten bei der letzten LED starten und zurücklaufen. Ergänze dein geschriebenes Programm.

9. Analoge Ausgabe - Dimmen einer LED

Bisher hast du die Pins 2-13 als ***digitale*** Ausgänge verwendet. Digitale Pins haben zwei Zustände.
Benenne die zwei Zustände:

Bezeichnung des Zustands:	Anweisung zum Aufrufen des Zustands	Wert des elektrischen Potenzials:

Manchmal möchte man ein Gerät nicht nur an- und ausschalten, sondern in vielen Zwischenschritten ansteuern.
Sogenannte ***PWM Pins*** (PWM von ***P**uls**w**eiten**m**odulation*) können 256 verschiedene Zustände einnehmen. Das heißt, dass die maximale Leistung eines PWM Pins (beginnend bei Null) in 256 Schritten erreicht wird.
Da die PWM Pins auch als digitale Ausgänge verwendet werden können, findest du sie auf der Seite der digitalen PINs.

Welche der Pins von 2-13 sind **PWM** Pins? _____

Beispiel:

Dimmen_einer_LED

```
/* Dimmen einer LED
Nick Tech
20.02.2013
*/
int LED = 9;                    // PWM Pin 9 für LED
void setup()
{}
void loop()
{
  analogWrite(LED, 255);        // PWM-Wert zwischen 0-255
}
```

Tipp: Die PWM-Funktion muss nicht im setup() definiert werden.

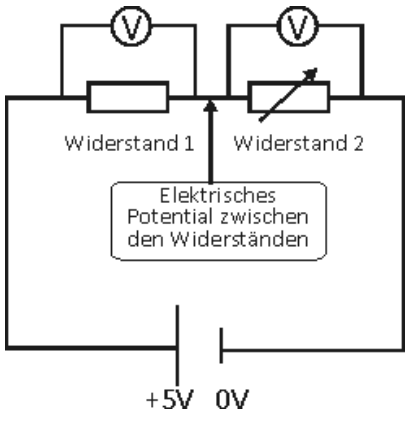
Aufgabe 9.1: Teste das Programm mit verschiedenen Werten zwischen 0 und 255.

Aufgabe 9.2: Verwende eine Zählschleife um die Helligkeit der LED von 0 auf 255 zu steigern.
Tipp: Bei einer Zählschleife bedeutet die Laufbedingung x++ eigentlich x=x+1. Du kannst gerne mit anderen Laufbedingungen experimentieren.

Aufgabe 9.3: Kannst du die LED auch wieder langsam abdunkeln?

10. Abfrage von Sensoren - Der Arduino als Messgerät

Bislang hast du die Pins des Arduino immer als ***Ausgang*** benutzt, das heißt, du hast ein Potenzial von +5V oder von 0V angelegt. Nun sollen die Pins als ***Eingang*** verwendet werden, der Arduino soll also messen, welches Potenzial an einem bestimmten Pin anliegt.

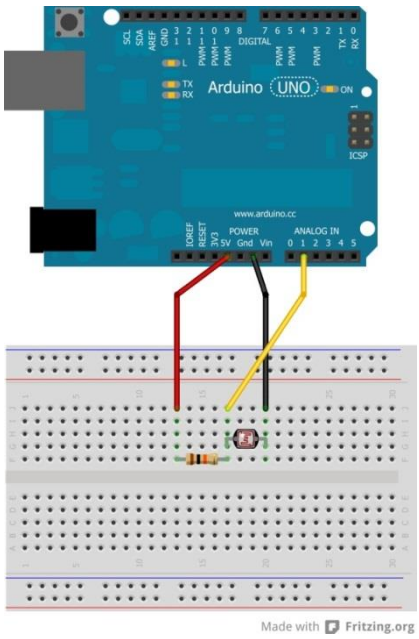


Helligkeitsmessung mit dem LDR

Als erstes soll der Arduino einen lichtabhängigen Widerstand (LDR) auslesen – dieser kann als Helligkeitssensor verwendet werden.

Um das Potenzial zu verändern, muss man einen Spannungsteiler aufbauen (siehe Abbildung rechts).

Diese Schaltung sieht am Arduino so aus.



- Festwiderstand an +5V.
- Der LDR an Gnd.
- Jeweils ein Anschluss der beiden Widerstände wird mit Analog-Pin 1 verbunden.

Ausgelesen wird der Wert mit dem Befehl ***analogRead():***

Auslesen_eines_LDR_ino

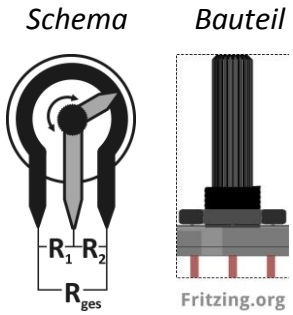
```
int LDR= 1;           // Sensoreingang
int Helligkeit;       // Wert des LDR

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

void loop()
{
  Helligkeit = analogRead(LDR); //Auslesen und Zuordnung zur Variablen
  Serial.println(Helligkeit);
  delay (200);
}
```

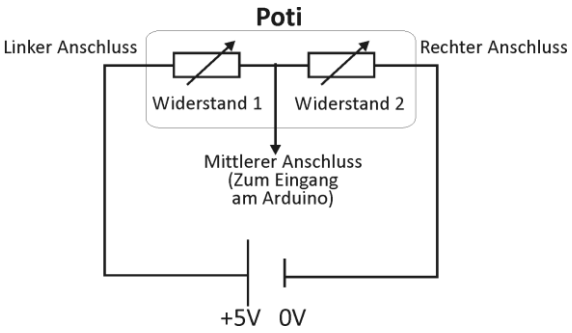
Aufgabe 10.1: Erstelle die Schaltung, und das Programm und lass dir den Helligkeitswert am Bildschirm ausgeben.

Aufgabe 10.2: Erstelle einen einfachen Helligkeitsanzeiger: Je nach Helligkeit soll eine LED schneller oder langsamer blinken.

Ein Potentiometer als Drehregler

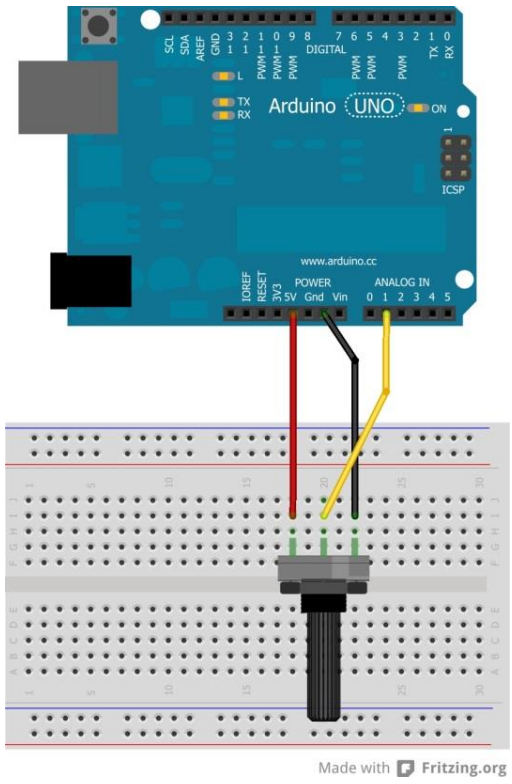
Ein Potentiometer (kurz Poti) ist ein regelbarer Widerstand. Zwischen den beiden äußeren Anschlüssen herrscht ein konstanter Widerstand (z.B. R_{Ges}= 10kΩ). Über den Drehknopf kann der Widerstand zwischen dem mittleren Anschluss und den beiden äußeren Anschlüssen verändert werden. Der Gesamtwiderstand wird hierbei aufgeteilt (z.B. R_1=7 kΩ, R_2=3 kΩ). Dadurch besteht das Poti im Prinzip also aus zwei regelbaren Widerständen in einem Bauteil.





Wir können also unseren Spannungsteiler mit nur einem Bauteil erhalten!

Das vereinfacht den Anschluss an den Arduino:



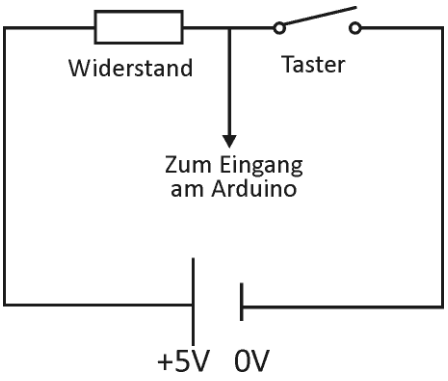
Aufgabe 10.3: **Erstelle die Schaltung und benutze das Programm aus Aufgabe 1, um dir den Wert des Poti am Bildschirm auszugeben.**

Aufgabe 10.4: **Ändere die Variablen (LDR→ Sensor / Helligkeit → Sensorwert) und speichere das Programm unter „Sensorabfrage“. So kannst du es in Zukunft als Vorlage für alle Sensorabfragen verwenden.**

Aufgabe 10.5: **Benutze das Programm aus Aufgabe 2, um die Blinkfrequenz einer LED mit dem Poti zu steuern.**

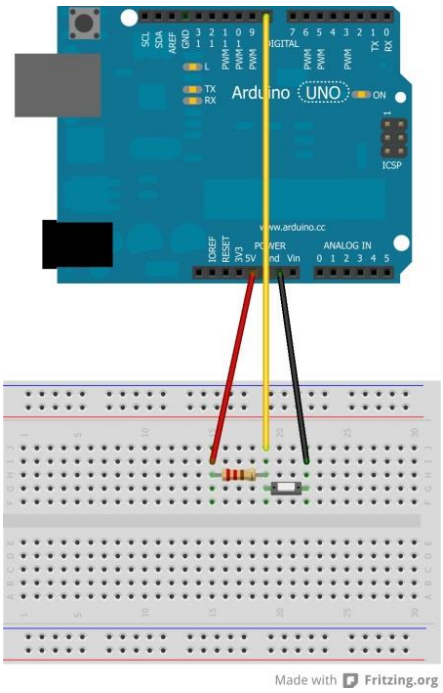
Ein Taster als „digitaler Spannungsteiler“

Ein Taster ist im Prinzip ebenfalls ein Sensor - auch er wird als Spannungsteiler aufgebaut. Das Besondere am Taster als Sensor ist, dass er nur zwei Zustände hat: Offen oder geschlossen. Das führt dazu, dass das Potenzial, das vom Arduino gemessen wird, nur zwei Werte annehmen kann: 0V oder +5V.



Zur Abfrage des Tasters wird daher nicht der Befehl ***analogRead*** sondern der Befehl ***digitalRead*** verwendet.

Der Vorteil hierbei ist, dass der Taster nicht an einen Analog-Eingang angeschlossen werden muss, sondern an einen der Digital-Pins des Arduino (2-13) angeschlossen werden kann.



Aufgabe 10.6: **Öffne dein Programm „Sensorabfrage“ und ändere es so ab, dass der Tasterzustand am Bildschirm ausgegeben wird.**
Welcher Wert wird bei gedrücktem Taster und welcher bei nicht gedrücktem Taster angezeigt?

Aufgabe 10.7: **Vertausche in deiner Schaltung die Position des Tasters mit der des Widerstandes und lass dir erneut die Werte am Bildschirm ausgeben.**

Info: -Je nach Position des Widerstandes wird er als ***Pull-up*** oder als ***Pull-down***-Widerstand bezeichnet. Ein Pull-up-Widerstand „zieht“ das Potential bei Drücken des Tasters „nach oben“ (Änderung von 0V auf +5V).

11. Die „If“-Bedingung - Der Arduino reagiert auf seine Umwelt

Bis jetzt hast du gelernt, wie der Arduino Ausgänge steuern und wie er Sensorwerte auslesen kann. Nun soll beides miteinander verknüpft werden.

Mit der ***If*** – Bedingung kann man den Arduino dazu bringen, eine *Anweisung* auszuführen, *wenn* eine bestimmte *Bedingung* erfüllt ist, z.B soll er eine LED anschalten, wenn die Helligkeit der Umgebung abnimmt. Die *Bedingung* wäre hierbei also, dass die Helligkeit unter einen bestimmten Wert fällt, die *Anweisung* wäre, die LED anzuschalten.

Das Programm „wenn die Helligkeit unter 300 ist, dann schalte die LED an, ansonsten schalte sie aus“ heißt in der Sprache des Arduino:

```
void loop()
{
 helligkeit = analogRead(ldr); //Sensor wird ausgelesen und der Variablen helligkeit zugewiesen
 if (Helligkeit<300)           //Bedingung
   {                           //Beginn der Anweisung - kann beliebig lang sein
    digitalWrite(LED,HIGH);    //LED anschalten
   }                           //Ende der Anweisung
 else                          //Ansonsten (wenn die Bedingung nicht erfüllt ist)
   {
     digitalWrite(LED,LOW);     //LED ausschalten
   }
}
```

Allgemein heißt das:

if (*Bedingung*)
{ *Anweisung* 1; }
else
{ *Anweisung* 2; }

Aufgabe 11.1: Schreibe das Programm, das die beschriebene Funktion hat und erstelle die zugehörige Schaltung (Tipp: nutze dazu deine bereits vorhandenen Programme).

Aufgabe 11.2: Verändere deine Ampelschaltung so, dass die Ampel bei Tagbetrieb normal funktioniert, nach Einbruch der Dunkelheit aber nur noch die gelbe LED blinkt (Nachtbetrieb).

Aufgabe 11.3: **Die meisten Ampeln haben auch noch eine Fußgängerampel. Erstelle eine Schaltung für eine Fußgängerampel mit „Grünanforderung“. Wenn der Fußgänger auf einen Knopf drückt, schaltet die Autofahrerampel auf rot und die Fußgängerampel auf grün. Nach 3 Sekunden schaltet die Fußgängerampel wieder auf rot und die Autos dürfen wieder fahren.**
Schreibe das entsprechende Programm.

12. Unterprogramme - Programmieren mit Bausteinen

Deine beiden Ampelprogramme der letzten beiden Aufgaben sollen kombiniert werden. Die Fußgängerampel mit dem Tag-und Nachtbetrieb.

Ganz schnell wird das Programm sehr unübersichtlich. Um es einfacher zu gestalten kann man einzelne „Programmbausteine“ (Methoden) einsetzen. Das Blinken für den Nachtbetrieb gibt einen eigenen Programmbaustein und der Tagbetrieb einen anderen. Diese Programmbausteine werden als „void“ definiert und zwischen „void setup()“ und „void loop()“ geschrieben. Aufgerufen werden sie dann im Hauptprogramm an der Stelle, an der das Unterprogramm ausgeführt werden soll.

Das Beispiel zeigt eine Alarmanlage. Wird ein Taster gedrückt, soll eine LED für eine Sekunde aufleuchten. Dieses Aufleuchten wird als Unterprogramm „Warnblink()“ geschrieben, der Aufruf erfolgt im Hauptprogramm (mit Strichpunkt, da es sich um eine Anweisung handelt).

Unterprogramm_ino §

```
int LED;
int Button=6;
int Buttonstate;

void setup()
{
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  }

void Warnblink()
{
  digitalWrite(LED, HIGH);
  delay(1000);
  digitalWrite(LED, LOW);
  delay(1000);
 }

void loop()
{
  Buttonstate=digitalRead(Button);
  if (Buttonstate==HIGH)
  {
   Warnblink();
  }
  else
  {}
  }
```

Allgemein heißt das:

```
void  Unterprogramm ()
  {
Anweisung 1;
  }
 void loop ()
{
Unterprogramm();
 }
```

Aufgabe 12.1: Kombiniere die Programme für die Fußgängerampel (Tagbetrieb) mit dem Programm, das zwischen Tag- und Nachtbetrieb wechselt. Definiere dazu die entsprechenden Unterprogramme und kopiere die dazugehörigen Befehle hinein.

Unterprogramme für Profis

Bisher stehen die Unterprogramme für sich und können nur vom Hauptprogramm aus als unveränderlicher Baustein aufgerufen werden. Es gibt allerdings auch die Möglichkeit, Werte vom Hauptprogramm in das Unterprogramm hineinzugeben.

Im nebenstehenden Beispiel wird das Unterprogramm 2mal nacheinander aufgerufen. Beim ersten Mal wird die rote LED angeschaltet, beim zweiten Mal die gelbe.

Außerdem wird definiert, dass das Unterprogramm einen Wert aus dem Hauptprogramm übernimmt (im ersten Fall 500, im zweiten Fall 1000) und diesen der Variable Zeit im Unterprogramm zuordnet.

Aufgabe 12.2: Beschreibe genau, was das dargestellte Programm macht.

Unterprogramm2

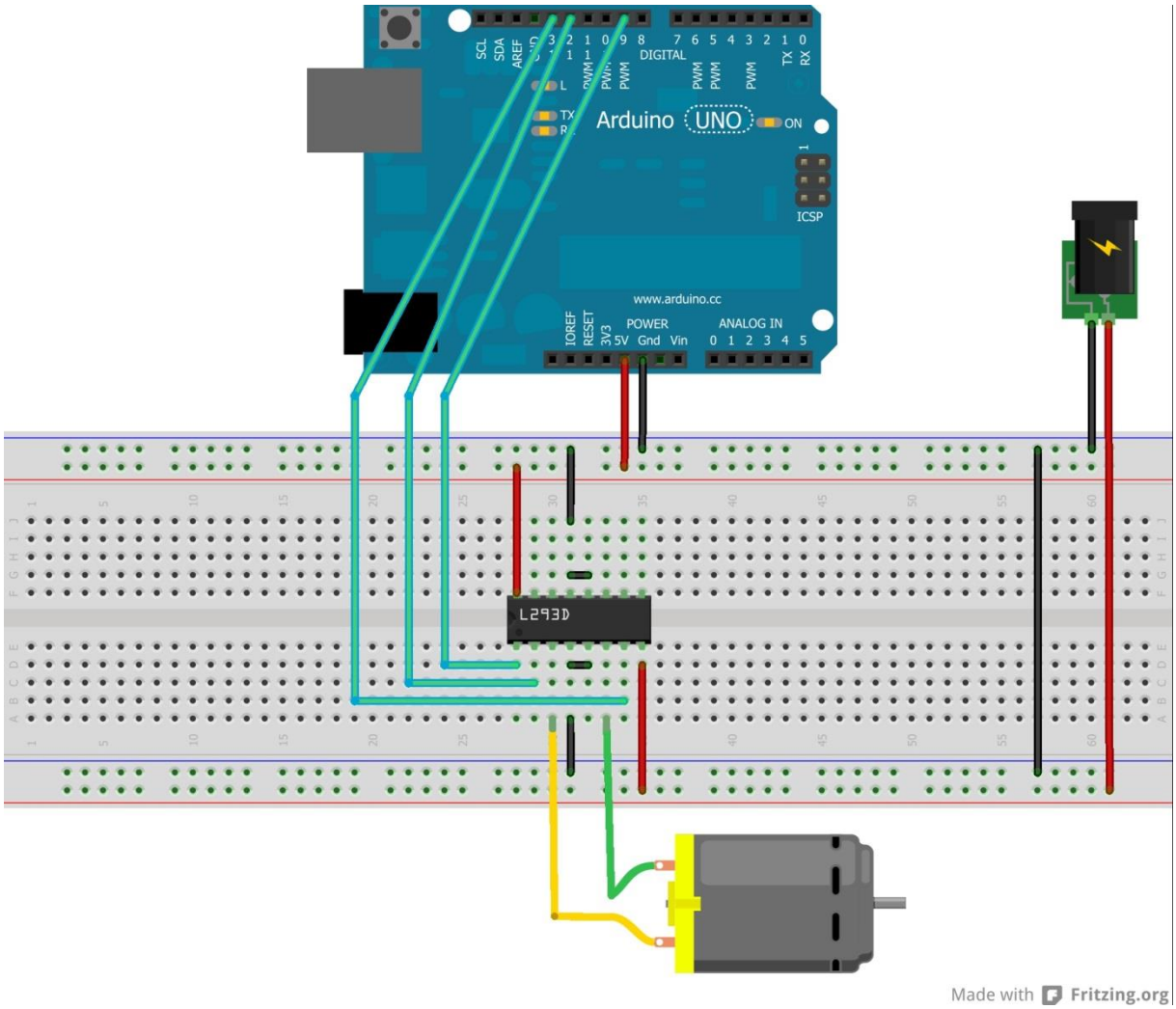
```
int LED;
int Zeit;
int rot=5;
int gelb=6;

void setup()
{
  pinMode(rot, OUTPUT);
  pinMode(gelb, OUTPUT);
  }

void Warnblink(int Zeit)
{
  digitalWrite(LED, HIGH);
  delay(Zeit);
  digitalWrite(LED, LOW);
  delay(2*Zeit);
 }

void loop()
{
  LED=rot;
  Warnblink(500);
  LED=gelb;
  Warnblink(1000);
}
```

13. Motorsteuerung - Endlich kommt Bewegung in die Sache…

Bislang haben wir eigentlich LEDs zum Leuchten gebracht. Nun soll Bewegung in die Sache kommen – wir wollen Motoren ansteuern.

Allerdings liefert der Arduino eine Stromstärke von **maximal 40 mA** – für einen ordentlichen Motor ist das viel zu wenig. Deshalb benutzt man einen zusätzlichen Chip, einen sogenannten Motortreiber. Der Motortreiber, den wir benutzen heißt **L293D**. An ihn können bis zu zwei Motoren angeschlossen werden, die sich in jeweils vor- und rückwärts drehen lassen.

Um den Motor zu steuern müssen wir an den Motortreiber ein Netzgerät (für die Energieversorgung des Motors) den Arduino (für die Steuerung) und natürlich den Motor selbst anschließen:



Funktion des L293D

Drehzahl des Motors (0-255) – Anschluss an PWM-Ausgang des Arduino (analogWrite(drehzahl,200);)

Anschalten des Motors – Anschluss an Digital-Ausgang des Arduino. Wird dieser Pin HIGH geschaltet, so wird am Motoranschluss OUT1 das Potenzial des Netzgerätes angelegt. Wird dieser Pin LOW geschaltet, so wird am Motoranschluss OUT1 0V angelegt.

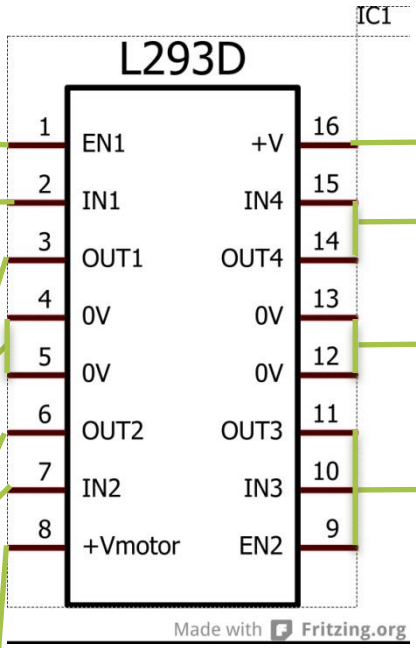
Motoranschluss OUT1

Ground – **Alle** Grounds müssen mit einander verbunden werden – auch mit dem Netzgerät und dem Arduino!

Motoranschluss OUT2

Anschalten des Motors – Anschluss an einen zweiten Digital-Ausgang des Arduino. Wird dieser Pin HIGH geschaltet, so wird am Motoranschluss OUT2 das Potenzial des Netzgerätes angelegt. Wird dieser Pin LOW geschaltet, so wird am Motoranschluss OUT2 0V angelegt.

Energieversorgung des Motors – Anschluss an den Pluspol des Netzgerätes.



Stromversorgung des L293D – Anschluss an +5V am Arduino

Für den zweiten Motor

Ground – Auch diese Grounds müssen mit verbunden werden.

Für den zweiten Motor

Aufgabe 13.1: Baue die Schaltung entsprechend der ersten Abbildung auf!

Aufgabe 13.2: Lass den Motor mit wechselnder Geschwindigkeit laufen.

Tipp: Bei schnell und ständig wechselnder Geschwindigkeit ist die elektrische Stromstärke durch den Motorchip sehr hoch. Das kann zu einer Überhitzung des L293D führen!

Aufgabe 13.3: Steigere die Motorgeschwindigkeit mit Hilfe einer for-Schleife.

14. Hast du Töne - Lautsprecher am Arduino betreiben

Mit dem Arduino kann man auch Töne und sogar Melodien an einem kleinen Lautsprecher ausgeben. Es gibt mehrere Möglichkeiten um Töne zu erzeugen. Zwei Methoden seien hier genannt. Diese Methoden sind in erster Linie dazu geeignet, um Quittungstöne zu erzeugen oder einfache Melodien zu spielen:

Lautsprecher

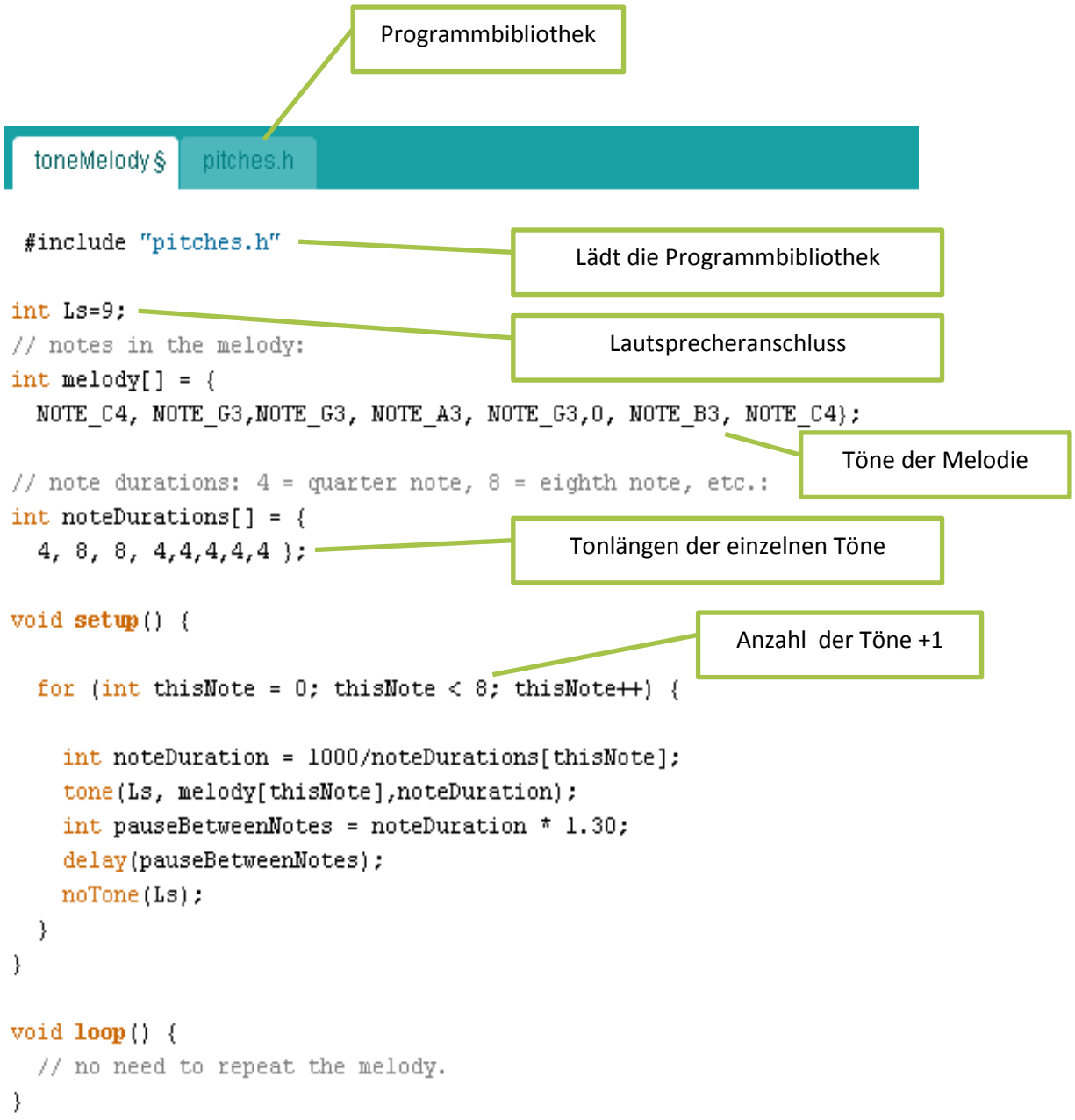
```
int LS=9;                // Lautsprecher an Pin 9 und GND anschließen

void setup()
{
pinMode(LS, OUTPUT);     // der Pin9 wird als Ausgang definiert
}
void loop()
{
digitalWrite(LS, HIGH);
delay(1);                // wartet 1 Millisekunde
digitalWrite(LS, LOW);
delay(1);                // wartet 1 Millisekunde

// ein Periode dauert T=2ms d.h. f=500Hz
}
```

Grundsätzlich gilt, um einen Ton zu erzeugen, wird der Lautsprecher immer wieder ein- und ausgeschaltet. Die Tonhöhe wird durch die Frequenz bestimmt: Ein Ton, mit einer Frequenz von 500Hz wird erzeugt, wenn der Lautsprecher für 1 ms ein- und 1 ms ausgeschaltet wird.

Für höhere Frequenzen kann die Anweisung *delayMicroseconds (x);* verwendet werden.

Auf diese Art und Weise eine Melodie zu spielen ist sehr aufwändig. Für jeden einzelnen Ton muss die Frequenz bestimmt werden und in die Zeitdauer, die der Lautsprecher warten soll, umgerechnet werden. Genau diese Umrechnung hat ein freundlicher Programmierer bereits erledigt und in einer eigenen „Programmbibliothek“ gespeichert. Für uns heißt das, dass wir diese Programmbibliothek einfach benutzen können und somit eine Melodie mit echten Notennamen schreiben.

Im Folgenden ist ein einfaches Programm dargestellt, das man auf seine Melodie anpassen kann.




```
#include "pitches.h"

int Ls=9;
// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 8, 8, 4,4,4,4,4 };

void setup() {

  for (int thisNote = 0; thisNote < 8; thisNote++) {

    int noteDuration = 1000/noteDurations[thisNote];
    tone(Ls, melody[thisNote],noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    noTone(Ls);
  }
}

void loop() {
  // no need to repeat the melody.
}
```

Ein ähnliches Programm findest du im Internet unter folgendem Link:
http://arduino.cc/en/Tutorial/Tone
von dieser Seite aus, kannst du dieses Programm direkt in dein Sketch kopieren und entsprechend verändern (das spart viel Arbeit).

Dort findest du auch das Programm **pitches.h,** welche du ebenfalls noch einfügen musst.

1. Kopiere die Datei pitches.h.
2. Gehe zurück zu deinem Sketch und erstelle einen neuen Tab, indem du in der rechten oberen Ecke auf folgenden Button drückst:
3. Erstelle einen neuen Tab und gebe der neuen Datei den Namen **pitches.h** .
4. Füge die kopierte Datei ein. Fertig!

Diese Seiten beinhalten übrigens einige sehr wertvolle Informationen, die über das Gesagte im Unterricht weit hinausgehen. Vielleicht findest du dort auch Lösungen für andere Probleme, die du beim Programmieren bisher noch nicht lösen konntest.

15. Der richtige Winkel - Servos am Arduino betreiben

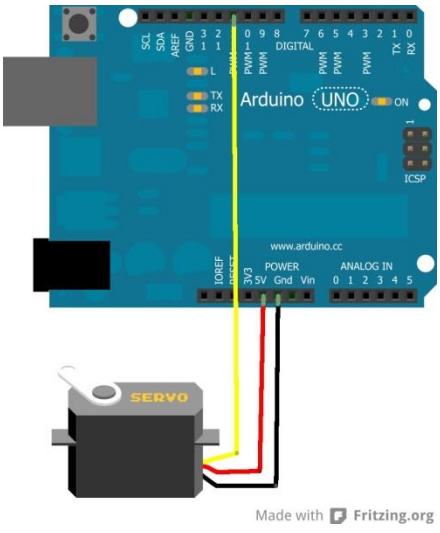
Ein Servo bezeichnet in der Elektrotechnik einen Verbund aus Ansteuerungs- und Antriebseinheit. Dies kann beispielsweise ein Elektromotor samt seiner Steuerelektronik sein (Wikipedia). Im Modellbau und beim Roboterbau werden häufig Servos benutzt, da man mit diesen ganz einfach Arme oder Höhenruder in ganz bestimmte Positionen fahren kann, denn einem Servo kann man exakt vorgeben, welchen Winkel er anfahren soll.

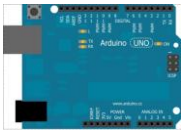
Ein Servo hat üblicherweise 3 Anschlüsse:

Rot:	+5V
Braun oder Schwarz:	Gnd
Orange oder gelb:	Steuerungsleitung

Die Steuerungsleitung muss an einen PWM-Ausgang angeschlossen werden (in unserem Beispiel Pin 11).

Für den Arduino gibt es nun eine Programmbibliothek, die die komplizierte Ansteuerung der Steuerelektronik des Servos übernimmt.





Servo §

```
#include <Servo.h>       // lädt die Programmbibliothek

Servo meinservo;         // benennt den Servo ("ein Objekt wird kreiert")

void setup()
{
  meinservo.attach(11);  // Pin, an dem der Servo angeschlossen ist.
}

void loop()
{
  meinservo.write(0);    // Fährt den Servo auf Position 0
  delay(200);
  meinservo.write(90);   // Fährt den Servo auf einen Winkel von 90°
  delay(200);
}
```

Aufgabe 15.1: **Baue die Schaltung entsprechend der Abbildung auf und lasse einen Servo verschiedene Winkel anfahren.**

Aufgabe 15.2: **Schreibe ein Programm für eine „Fernsteuerung“ – mit einem Drehpotentiometer soll die Position des Servos eingestellt werden.**

Notizen

Unterrichtsskript zum Thema Steuern und Regeln im NwT-Unterricht
1. Auflage, 2013, Friedrich-Schiller-Gymnasium, Marbach a. Neckar
Autoren: StD Martin Merkle und StD Frank Trittler