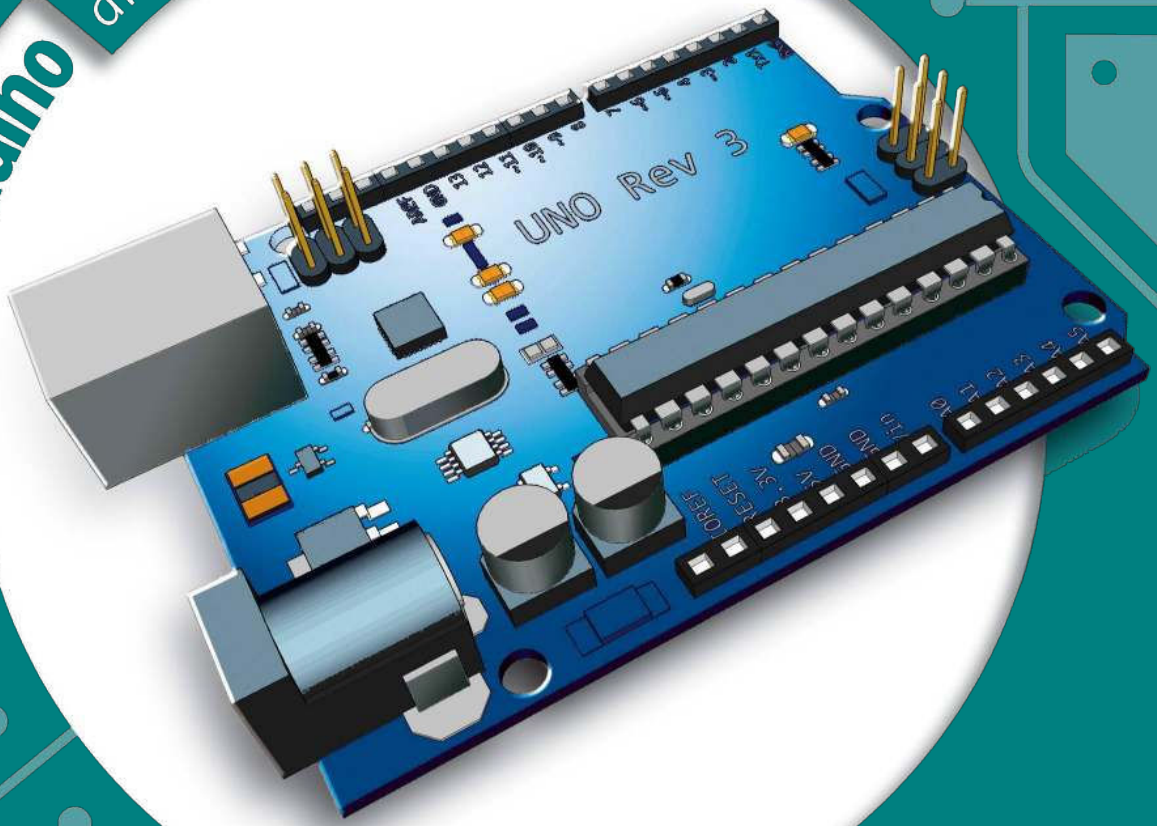
Einführung in den Mikrocontroller Arduino

Der **Arduino** als Steuerzentrale



Version 1.0

Inhaltsverzeichnis

Was ist ein **Mikrocontroller ?**

1

Mikrocontroller sind kleine elektronische Bauteile, die in automatisch funktionierenden Geräten sozusagen das Gehirn darstellen, also die Steuerzentrale, die zum Beispiel...

..in einer Waschmaschine das Waschprogramm steuert und die Temperaturen regelt

..in einem Airbag entscheidet, ob ein Crash vorliegt und dann binnen 15 ms die Zündung auslöst

...eine Schiebetüre beim Herannahen von Personen automatisch öffnet

...oder in deinem nächsten Projekt Vorgänge steuert.

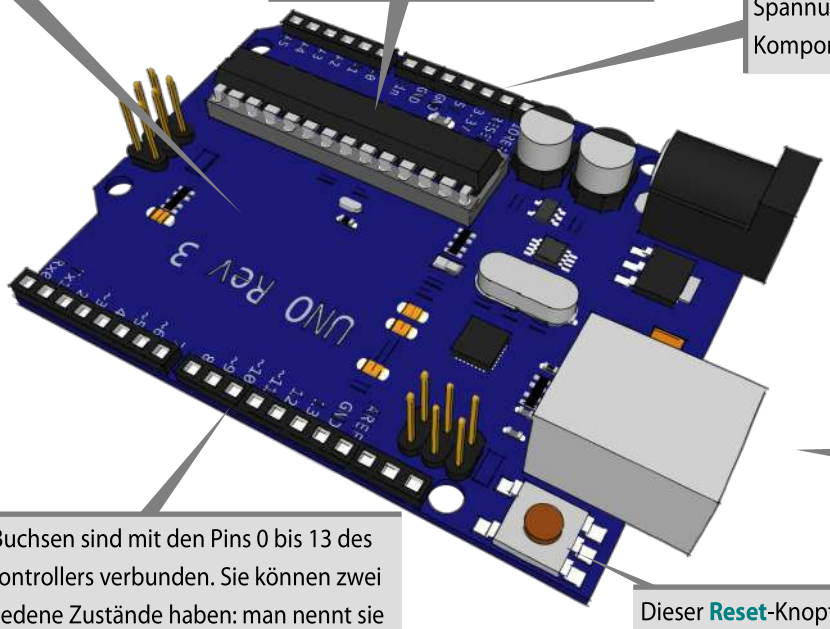
Mikrocontroller
Für das Wort Mikrocontroller verwendet man oft die Abkürzung µC. Das µ (sprich „mü“) ist ein griechischer Buchstabe und wird als Kürzel für „mikro“ verwendet.

Arduino
Arduino ist der Name einer einer Kneipe, in der sich die Erfinder des Arduino 2005 oft trafen.

Diese ganze Platine nennt man **Arduino**.

Das hier ist der eigentliche **Mikrocontroller** mit seinen vielen Anschlüssen, den sogenannten Pins.

Diese Steckanschlüsse (Buchsen) stellen Spannungen zum Anschluss anderer Komponenten zur Verfügung.



Hier kann man eine Batterie oder ein Netzteil anschließen, um den Arduino ohne USB-Anschluss zu betreiben.

Über den **USB-Anschluss** kann man Programme auf den Arduino übertragen.

Diese Buchsen sind mit den Pins 0 bis 13 des Mikrocontrollers verbunden. Sie können zwei verschiedene Zustände haben: man nennt sie **HIGH** und **LOW**.

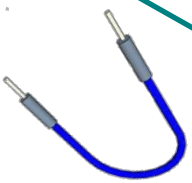
Dieser **Reset**-Knopf startet den Mikrocontroller neu.

2 Es leuchtet !

In diesem Kapitel lernst du die Komponenten und die Anschlüsse auf der Arduino-Platine kennen, mit denen du eine Leuchtdiode zum Leuchten bringen kannst:

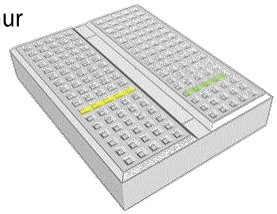
Eine **Leuchtdiode** (**LED**) könnt ihr euch zunächst wie eine Glühlampe vorstellen. Sie leitet Elektrizität aber nur vom längeren Anschlussdraht (**Anode** genannt) zum kürzeren Anschlussdraht (**Kathode**) und geht sehr schnell kaputt, wenn man sie ohne einen **Schutzwiderstand** einsetzt.

Ein **Widerstand** ist ein Bauteil, mit dem zum Beispiel Leuchtdioden geschützt werden können. Es leitet Elektrizität nur schlecht - je schlechter, desto höher sein so genannter Widerstandswert (angegeben in der Einheit Ohm Ω).

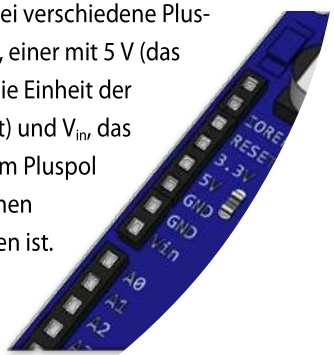


Für die Arbeit mit Breadboards gibt es sehr praktische Kabel, die sogenannten **Jump-Wires**. Hat ein solches Kabel zwei Stecker, bezeichnet man es als male-male, mit Stecker und Buchse als male-female und mit zwei Buchsen als female-female-Kabel.

Das **Bread board** (oder auch **Steckbrett**) dient dazu, Kabel ohne Löten elektrisch miteinander zu verbinden. Dazu sind jeweils 5 Buchsen in einer Reihe (siehe z.B. gelbe oder grüne Markierung) untereinander verbunden. Um zwei Kabel miteinander zu verbinden, muss man sie also nur in zwei Buchsen der selben Fünferreihe stecken.

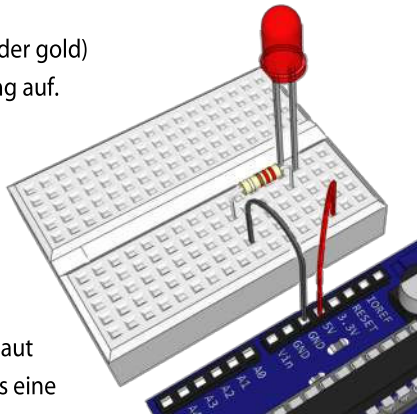


An dieser **Buchsenleiste** stellt der Arduino verschiedene elektrische Potentiale zur Verfügung. Die wichtigsten Anschlüsse sind drei Minuspole (mit **GND** beschriftet) sowie drei verschiedene Pluspole, einer mit 3,3 V, einer mit 5 V (das V steht jeweils für die Einheit der Spannung, das Volt) und V_{in}, das ganz direkt mit dem Pluspol der angeschlossenen Batterie verbunden ist.



Ganz wichtig: Niemals darf man beim Aufbau einer Schaltung eine direkte Verbindung zwischen einem Plus- und einem Minuspol erzeugen – das ist ein Kurzschluss, der den Arduino und in sehr ungeschickten Fällen auch den angeschlossenen Computer zerstören kann.

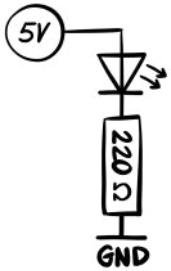
2.1 Baue aus einer LED, einem 220-Ohm-Widerstand (Farbkombination rot-rot-braun und dann silber oder gold) und ein paar Jump-Wires die abgebildete Schaltung auf.



Hinweis: In der Schaltung soll die Elektrizität vom Pluspol (5 V) durch die LED und den Schutzwiderstand zum Minuspol (GND) strömen. Achte also darauf, die LED richtig herum einzusetzen: Langer Anschluss am Pluspol.

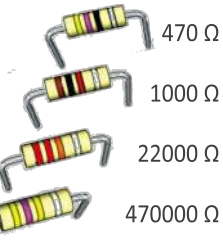
Ob der Widerstand vor oder hinter der LED eingebaut wird, spielt aber keine Rolle. Wichtig ist nur, dass es eine Reihenschaltung ist.

2.2



Techniker und Ingenieure würden die Schaltung als Schaltbild darstellen. In einem **Schaltbild** hat jedes Bauteil ein (üblicherweise europa- oder sogar weltweit) festgelegtes Schaltsymbol. Lege in deinem Heft eine Übersichtsseite an, auf der du nach und nach alle Schaltsymbole und deren Bedeutung auflistest. Die ersten drei Symbole (für LED, Widerstand und Minuspol) kannst du herausfinden, indem du dieses Schaltbild mit deiner Schaltung vergleichst.

2.3 Aus Informatik in Klasse 7 kennst du bereits Codierungen - die Werte von Widerständen sind als **Farbringe** codiert. Kannst du den Code an den Beispielen rechts erknobeln?



schwarz	0
braun	1
rot	2
orange	3
gelb	4
grün	5
blau	6
lila	7
grau	8
weiß	9

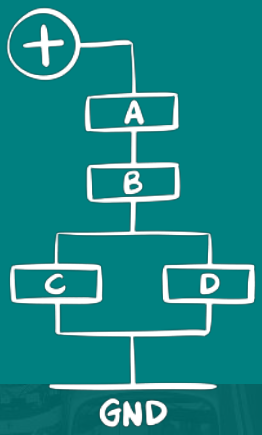
Zwei Tipps gibt es dazu: Die Farbtabelle ist nur ein Teil der Codierung. Und: man beginnt den Code immer auf der Seite zu lesen, auf der die Ringe einen engen Abstand haben. Der vierte Ring, meistens silbern oder goldfarben, gibt an, wie genau der Widerstand ist. Bei Gold sind es 5%, bei Silber 10% und 20% sind es, wenn der vierte sog. **Toleranz-Ring** ganz fehlt.

Welchen Wert hätte ein Widerstand mit den Ringfarben grün—grün—grün?

Reihenschaltung und Parallelschaltung

Bauteile können in einem Stromkreis als Reihenschaltung oder als Parallelschaltung angeordnet werden. Zwei Bauteile sind in Reihe schaltet, wenn Elektrizität erst durch das eine und dann durch das andere Bauteil fließt. Bei einer Parallelschaltung gibt es mehrere Wege für die Elektrizität.

Im Beispiel sind die Bauteile A und B „in Reihe geschaltet", C und D „parallel".



3 Programm zum **Programmieren**

Programme für Mikrocontroller schreibt man in sogenannten **Entwicklungsumgebungen** (**IDE** = integrated development enviroment) auf einem Computer und überträgt sie dann z.B. per USB-Schnittstelle. Die Entwicklungsumgebung für die vielen verschiedenen Arduino-Mikrocontroller heißt **arduino.exe** und sieht aus wie ein ganz normales Programm mit einigen Sonderfunktionen:

In diesem großen Feld schreibt man das Programm, das beim Arduino auch als **Sketch** bezeichnet wird...

...und mit diesem Button kannst du es probeweise kompilieren (siehe links) und so auf Schreibfehler prüfen.

Mit einem Klick auf den Pfeil wird das Programm per USB auf den Arduino übertragen.

In vielen Programmiersprachen werden Programme in „Anweisungsblöcke“ strukturiert. Ein **Block** besteht hier aus einem Paar **geschweifter Klammern** (AltGr+7 bzw. 0) sowie den Zeilen dazwischen.

Hier werden Hinweise zu auftretenden Fehlern angezeigt.

Zwei Anweisungsblöcke sind vorgeschrieben: Die Anweisungen im Block **setup()**{…} werden direkt ausgeführt, sobald man den Arduino startet. Danach werden die Anweisungen im Block **loop()**{…} ausgeführt und immer wieder wiederholt…Das nennt man auch „Endlosschleife“.

Alles was in einer Zeile hinter zwei Schrägstrichen („**//**“, **slash**) steht, gilt als **Kommentarzeile** und wird vom µC ignoriert. Dieses Programm enthält also keine einzige echte Anweisung, sondern nur zwei leere Blöcke.

Wo bekommt man die Entwicklungsumgebung?
Für zuhause kannst du dir die Entwicklungsumgebung von der offiziellen Seite des Projekts arduino.cc herunter laden. Sie ist dort für Linux, Windows und OS X erhältlich.

Die Arduino-Software ist sogenannte **Open Source Software** . An Open Source Software darf jeder einfach weiter programmieren, kann sie weiter verbreiten oder sogar weiter verkaufen. Mehr über die Ideen hinter Open Source findest du auch unter www.opensource.org oder dem QR-Code.

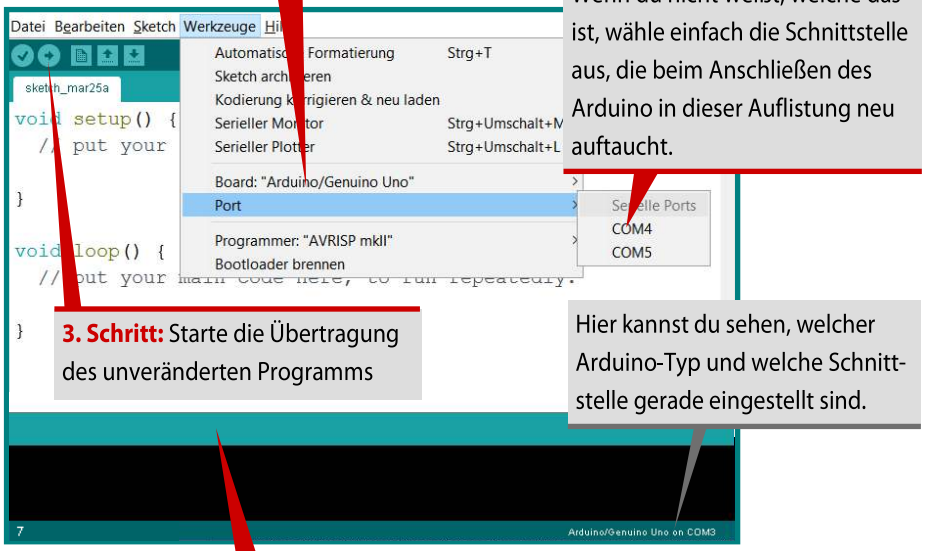
Kompilieren
Bevor das Programm auf den Arduino überspielt werden kann, wird es aus der für Menschen gut lesbaren Programmiersprache in eine für den Mikrocontroller gut lesbare und sehr kompakte Maschinensprache übersetzt. Diesen Vorgang nennt man „kompilieren“.

3.1 In dieser Aufgabe lernst du, wie das unveränderte „leere“ Programm auf den Arduino übertragen wird. Da die Entwicklungsumgebung für viele verschiedene Arduino-Boards geeignet ist, die an allen möglichen Schnittstellen des Computers angeschlossen werden. Deshalb sind erst einmal **zwei Einstellungen** notwendig:

1. Einstellung: Wähle hier den richtigen Arduino-Typ aus. Dein „Board“ heiß „Arduino Uno“.

2. Einstellung: Wähle die richtige Schnittstelle (engl. Port) aus.

Wenn du nicht weißt, welche das ist, wähle einfach die Schnittstelle aus, die beim Anschließen des Arduino in dieser Auflistung neu auftaucht.



3. Schritt: Starte die Übertragung des unveränderten Programms

Hier kannst du sehen, welcher Arduino-Typ und welche Schnittstelle gerade eingestellt sind.

Nach der erfolgreichen Übertragung steht in der Statuszeile so etwas wie „Upload done“, „Upload abgeschlossen“ oder „Hochladen abgeschlossen“. Wenn es nicht funktionierte, steht hier „**Upload failed**“ oder „**Probleme beim Hochladen**“. Dann solltest du die Einstellungen noch einmal prüfen.

3.2 Finde heraus, was die Zahl „7“ unten links bedeutet. Bei dir kann hier auch eine andere Zahl stehen. Es kann auch sein, dass hier keine Zahl steht...dann musst du erst herausfinden, wann sie erscheint.

Wenn beim Anschließen keine Schnittstelle hinzukommt?
Viele preisgünstige Arduino-Boards benötigen einen chinesischen USB-**Treiber**, den Windows und OSX nicht automatisch kennen. Du kannst ihn auf der Seite des chinesischen Herstellers *http://www.wch.cn/download/CH341SER_EXE.html* herunter laden und dann starten.

Fehlertypen:
Es gibt verschiedene Arten von Fehlern, die beim Programmieren auftreten können: Schreibfehler (sog. **Syntaxfehler**) führen dazu, dass das Programm nicht in Maschinencode kompiliert werden kann. **Übertragungsfehler** bewirken, dass das Programm nicht auf dem Arduino ankommt. **Laufzeitfehler** treten auf, wenn man beim Programmieren inhaltlich Fehler gemacht hat und das Programm nun auf dem Arduino etwas Falsches macht.

4 Das erste **Blinken**

In diesem Kapitel schreibst du dein erstes eigenes Programm und bringst damit eine LED zum Blinken. Dazu wird die LED mit ihrer Anode (längerer Anschluss) nicht mehr fest an 5 V oder 3,3 V angeschlossen, sondern an einen Pin des Mikrocontrollers (in der Abbildung ist es Pin 13). Das unten stehende Programm schaltet diesen Pin dann abwechselnd an (5 V) und aus (0 V) - man sagt dazu auch „HIGH“ und „LOW“. Wenn er „HIGH“ ist, fließt Elektrizität aus dem Pin durch die LED und den Widerstand zum Minuspol (GND). Wenn er LOW ist, fließt nichts... und die Leuchtdiode ist aus.

So sieht das Programm dazu aus:

```
void setup(){
 pinMode(13, OUTPUT);
}

void loop(){
 digitalWrite(13,HIGH);
 delay(1000);
 digitalWrite(13,LOW);
 delay(1000);
}
```

Damit die Leuchtdiode blinkt, wird Pin 13 hier „HIGH“ und wieder „LOW“ geschaltet.

Dazwischen wird jeweils 1000 Millisekunden lang gewartet.

Weil diese vier Anweisungen im Block loop() stehen, werden sie immer wieder ausgeführt.

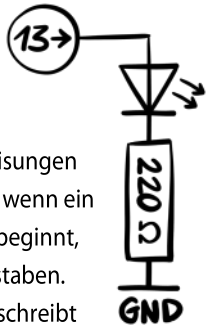
Damit Pin 13 überhaupt „HIGH“ oder „LOW“ geschaltet werden kann, muss es vorher in die Betriebsart Ausgang (**OUTPUT**) versetzt werden. Dazu dient die Anweisung pinMode(...). Das muss nur ein Mal erfolgen - deshalb steht diese Zeile im Unterprogramm „setup()“.

Digital?

Als digital (von lat. Digitus = Finger) bezeichnet man Daten, die nur wenige eindeutige Zustände haben können, so wie man mit Fingern nur 10 Zahlen zeigen kann. Bei digitalen Pins sind es sogar nur zwei Zustände: „HIGH“ und „LOW“.

Auch Geräte, die so arbeiten, bezeichnet man als digital.

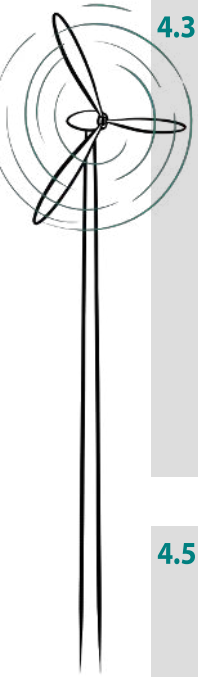
4.1 Schließe die LED nun an Pin 13 an und übertrage das Programm auf den Mikrocontroller.



Faustregel ist dabei: alle Anweisungen werden klein geschrieben, nur wenn ein neues Wort in der Anweisung beginnt, schreibt man einen Großbuchstaben. „HIGH“, „LOW“ und „OUTPUT“ schreibt man komplett in Großbuchstaben.

4.2 Ab welcher Dauer der Pausen kann dein Auge das Blinken nicht mehr erkennen?

Man sagt, Menschen hätten nur eine Erkennungsfrequenz von etwa 20 Hertz (Hz), könnten also nur etwa 20 Bilder pro Sekunde erkennen. Passt die von die gemessene Pausendauer zu diesem Wert?



4.3 Wie andere hohe Masten müssen auch Windkraftanlagen ein rotes Blinklicht an ihrer Spitze haben, um Flugzeuge zu warnen. Da die Spitze des Mastes bei einem Windrad aber nicht wirklich die Spitze der Gefahrenzone ist, blinkt ihr rotes Licht in einem besonderen Rhythmus: 1 Sekunde an, dann 0,5 Sekunden aus, dann wieder eine Sekunde an, dann aber 1,5 Sekunden aus.

Programmiere so ein Dauerblinklicht und übertrage das Programm. Läuft es?

4.4 Wie lange leuchtet in diesem Programm die LED an Pin 13 jeweils? Und wie lange ist sie aus?

```
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
}
void loop() {
  delay(400);
  digitalWrite(13, LOW);
  delay(200);
  digitalWrite(13, HIGH);
  delay(300);
}
```

4.5 Bei einem dieser Programme blinkt die LED nur ein Mal, beim anderen leuchtet sie nur dauerhaft. Welches ist welches? Und warum ist das so?

```
void setup(){
 pinMode(13, OUTPUT);
 digitalWrite(13,HIGH);
 delay(500);
 digitalWrite(13,LOW);
}
void loop() {
}
```

```
void setup(){
 pinMode(13, OUTPUT);
}
void loop() {
 digitalWrite(13,HIGH);
 delay(500);
 digitalWrite(13,LOW);
}
```

4.6 Besorge dir eine grüne, eine rote und eine gelbe LED sowie die Schutzwiderstände für 5 V und realisiere damit eine Ampel.

4.7 Lege dir auf einer Heftseite eine Liste an, in die du alle etwa 30 Anweisungen und Funktionen des Arduino mit einem Beispiel eintragen kannst. Bislang sind es nur fünf. Kannst du sie und ihre exakte Schreibweise schon auswendig?

Häufige Fehlermeldungen:

Die Entwicklungsumgebung zeigt dir teilweise auf deutsch und teilweise auf englisch an, wo sie einen Fehler vermutet. Hier einige häufige Fehler:

```
Problem beim Hochladen
```

Meistens hast du nicht die richtige Schnittstelle ausgewählt. Prüfe die Schritte von Seite 7.

```
expected ';' before…
```

Auf deutsch: ich erwarte einen Strichpunkt vor der Zeile, die markiert ist. Diese Strichpunkte vergessen viele Lernende am Anfang.

```
expected ')' before…
```

Es fehlt eine Klammer oder du hast einen Strichpunkt an Stelle eines Kommas gesetzt.

```
a function definition…
```

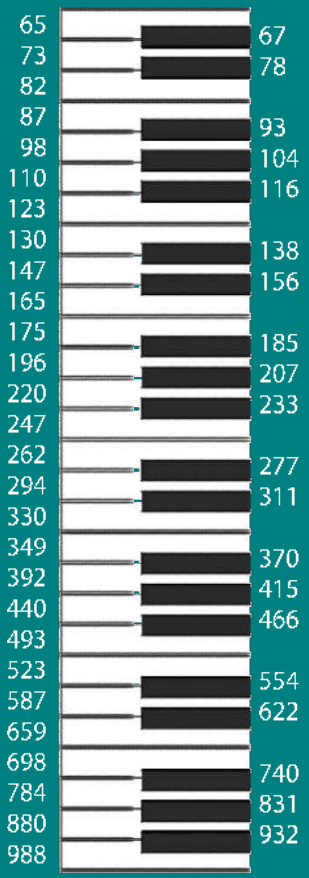
Prüfe die geschweiften Klammern.

```
…was not declared in
```

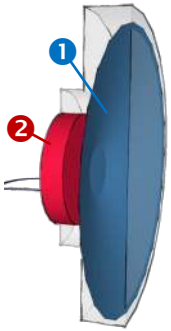
Der Compiler ist auf etwas Unbekanntes gestoßen. Wahrscheinlich hast du dich bei einer Anweisung vertippt.

5 Töne, Unterprogramme, Transistor

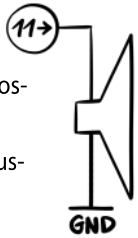
Menschen können Schallwellen mit Frequenzen zwischen etwa 18 und 18000 **Hertz** (1 **Hz** ist eine Schwingung pro Sekunde) hören. Welche Frequenz welcher Tonhöhe entspricht, zeigt die folgende Abbildung:



Hier lernst du, wie du mit dem Arduino und einem Lautsprecher Töne erzeugen und Tonfolgen abspielen kannst.



Viele Leute meinen, dass ein Lautsprecher sofort einen Ton von sich gebe, wenn er an eine Stromversorgung angeschlossen wird. Das ist aber falsch: Ein Lautsprecher besteht nämlich nur aus einer elastisch aufgehängten ❶ **Membran** und einem ❷ **Elektromagnet**. Fließt Elektrizität durch diesen Elektromagnet, wird die Membran angezogen, fließt keine Elektrizität, fällt die Membran wieder zurück. Ein Ton entsteht, indem die Membran schnell abwechselnd angezogen und wieder los gelassen wird, also die Stromversorgung abwechselnd ein– und ausgeschaltet wird.



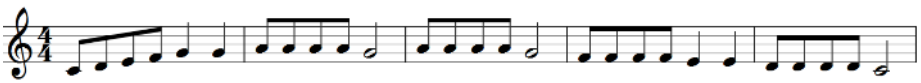
Um einen Ton zu erzeugen, kann der Lautsprecher also mit einem seiner Anschlüsse (egal welcher) zum Beispiel an Pin 11 und mit dem anderen an den Minuspol angeschlossen werden. Pin 11 muss schnell abwechselnd HIGH und LOW geschaltet werden. Weil man Töne häufig braucht, gibt es eine Anweisung, die sich um das schnelle Ein- und Ausschalten kümmert, selbst wenn der Mikrocontroller gerade etwas anderes macht:

```
void setup() {
}
void loop() {
 tone(11,440);
 delay(1000);
 noTone(11);
 delay(3000);
}
```

Diese Anweisung sorgt dafür, dass Pin 11 mit einer Frequenz von 440 Hz (Hertz) immer wieder HIGH- und LOW-geschaltet wird. Die Anweisung Tone sorgt übrigens vorher selbst dafür, dass Pin 11 in den pinMode OUTPUT geschaltet wird.

Diese Anweisung schaltet Pin 11 wieder dauerhaft auf LOW. Also ist der Ton aus.

5.1 Schließe einen Lautsprecher an einen Pin an und programmiere ein Lied. Wenn dir kein besseres einfällt, nimm „Alle meine Entchen“:



Hinweis: Wenn zwei Töne direkt auf einander folgen, brauchst du keine noTone-Anweisung dazwischen. Es geht also auch: tone(11,262); delay(500); tone (11,294); ….

Programmteile, die mehrfach benötigt werden (wie z. B. ein Refrain), muss man nicht mehrfach programmieren. Man legt dafür ein sogenanntes Unterprogramm an, das man wie eine ganz gewöhnliche Anweisung überall dort aufruft, wo es ausgeführt werden soll.

5.2

Vielleicht hat auch dein Musikstück einen Takt oder einen Refrain, der sich wiederholt? Probiere, diesen Teil als **Unterprogramm** zu schreiben:

```
void setup(){
}
void loop(){
  …
  refrain();
  …
  refrain();
  …
}
void refrain(){
  tone(11,220);
  delay(300);
  …
}
```

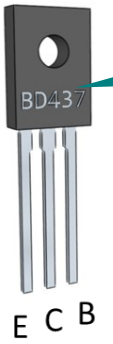
Am Ende fühlt sich ein Unterprogramm wie eine selbst geschriebene Anweisung an, hier zum Beispiel mit dem selbst gewählten Namen refrain(). Immer, wenn du refrain() schreibst, wird dein selbst definiertes Unterprogramm aufgerufen und ausgeführt.

Hier siehst du, wie das Unterprogramm mit dem Namen refrain() definiert wird. Es sieht genau so aus wie setup oder loop, hat aber einen selbst gewählten Namen und natürlich einen eigenen Zweck.

Wie viele Zeilen kannst du in deinem Musikstück durch Unterprogramme sparen?

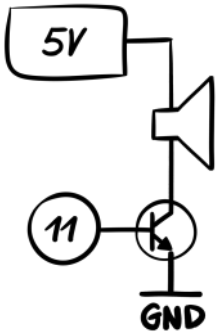
5.3

Dein Lautsprecher ist relativ leise, weil der Arduino durch seine Pins nur geringe elektrische Leistungen leiten kann. Höhere Leistung kannst du am Pin V_{IN} beziehen, das aber nicht programmierbar sondern einfach immer an ist. Mit einem **Transistor** kannst du trotzdem V_{IN} für den Lautsprecher nutzen:



Ein Transistor ist wie ein Schalter, der mit einem Pin gesteuert werden kann. Dieser Steuerpin wird an dies Basis (B) angeschlossen. Ist der Steuerpin HIGH, schließt sich der Schalter und lässt Elektrizität von C nach E durchfließen (es geht nur in dieser Richtung). Ist es LOW, fließt nichts.

Schließe Transistor und Lautsprecher so an, wie es das Schaltbild zeigt. Der Ton sollte nun lauter werden.



Namensgebung: Für die Namen von Unterprogrammen kann man beliebige selbst gewählte Worte verwenden, die keine Leerzeichen und keine Sonderzeichen (wie Umlaute oder Symbole) und als erstes Zeichen auch keine Zahl enthalten. Möglich wären also oma, opa oder hund16, unmöglich sind aber 16hund, rüpel oder a#. Dabei kommt es auf Groß– und Kleinschreibung an.

Natürlich darf man keine Bezeichner verwenden, die schon in der Programmiersprache verwendet werden.

Transistor: Die Anschlüsse des Transistors heißen **Basis**, **Kollektor** und **Emitter**. Der Transistor ist das technische Gerät, das am häufigsten auf der Erde hergestellt wird. Das liegt auch daran, dass alleine in einem modernen Computerprozessor mehr als 1 Milliarde Transistoren verbaut sind. Im Mikrocontroller auf dem Arduino sind es etwa 100000 - Transistoren können also sehr klein gebaut werden

6 Variablen

Variablen sind Speicherplätze, in denen sich der Mikrocontroller Zahlenwerte merken kann. Dabei gibt es kleine Speicherplätze, die nur ganze Zahlen von –32768 bis 32767 speichern können, große Speicherplätze, die ganze Zahlen von -2147483648 bis 2147483647 speichern können und solche Speicherplätze, die Kommazahlen speichern können. Das folgende Beispiel zeigt, wie man diese Speicherplätze **deklariert** (mit Namen versieht), wie man Zahlen darin speichert und gespeicherte wieder benutzt. Lies es aufmerksam durch, obwohl das Programm keinen echten Sinn hat:

```
int a, b, c;
long n, m;
float x, y;


void setup(){
  a=10;
  n=100-8;
  m=2*a+15-3;
  a=200+20;
}

void loop(){
  tone(11,a);
  delay(n);
  noTone();
  delay(m);
}
```

Hinter die Anweisung int kannst du Namen von Variablen schreiben, die Zahlen von –32768 bis 32767 speichern sollen. Diese Variablen nennt man auch „**Integer**“.

Hinter die Anweisung **long** schreibst du die Namen von Variablen für große Zahlen von –2147483648 bis 2147483647.

Diese Zeile macht x und y zu Variablen für Kommazahlen.

Das **„=“-Zeichen** bedeutet, dass der Wert rechts in der Variable links gespeichert wird. Der vorher gespeicherte Wert wird überschrieben.

Rechts vom Gleichheitszeichen kannst du auch Rechnungen hinschreiben. Diese werden dann berechnet und das Ergebnis in der Variable gespeichert. Rechenzeichen sind +, -, * und /.

Speichert man einen neuen Wert in eine Variable, wird der alte ersetzt.

Du kannst Variablen überall verwenden, wo sonst Zahlen stehen.

6.1 Welchen Wert haben die Variablen a, n und m am Ende des oberen Beispielprogramms?

6.2 Lege in deinem Musikstück (aus Aufgabe 5.2) Variablen für die Tondauern an, z.B. a für Achtelnoten, v für Viertel, h für Halbe und benutze sie in den Delays. So kannst du die Geschwindigkeit deines Stückes sehr leicht anpassen.

6.3 Was wird das Programm rechts wohl tun?

Probiere es aus!

```
int f;
void setup(){
  f=100;
}
void loop(){
  tone(11,f);
  delay(100);
  f=f+1;
}
```

Namensgebung: Für die Namen von Variablen kann man beliebige selbst gewählte Worte mit den gleichen Einschränkungen wie bei Unterprorammen verwenden.

Speicherplatz
Der Arduino hat in seinem Speicher insgesamt etwa 2000 Byte an Platz für Variablen. Integer benötigen jeweils 2 Byte, long oder float-Variablen jeweils 4 Byte.

Überlauf
Wenn man eine zu hohe Zahl in eine Variable speichern möchte, geschieht seltsames: In einem Integer ist 32767+1 zum Beispiel –32768. Diesen sogenannten Überlauf sollte man vermeiden.

Tipp:
Man kann Variablen bereits bei der Deklaration mit einem Wert versehen. Dazu schreibt man z.B.

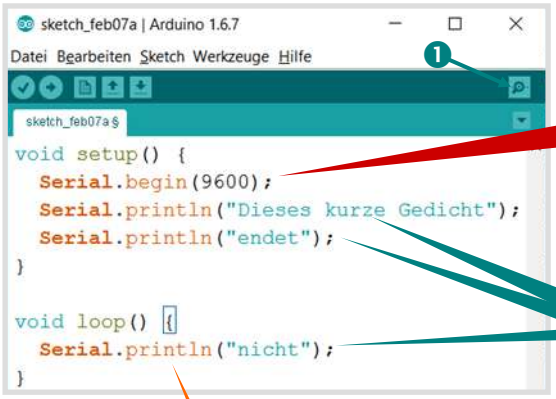
```
int a=8, b=-6;
long oma=87, p=99;
float haus=3.5;
```

Bei Kommazahlen schreibt man einen Dezimalpunkt.

Texte und Werte **anzeigen**

In diesem Kapitel geht es um den sogenannten „**seriellen Monitor**“. Das ist ein Programmfenster auf dem Computer, das Nachrichten anzeigt, die der Arduino verschickt. Du kannst dieses Programmfenster mit einem Klick auf die ❶ Lupe öffnen.

Dieses Beispiel zeigt dir, wie man den Arduino programmiert, damit er Texte verschickt:



Mit dieser Zeile wird die Verbindung zum Computer hergestellt - man sollte das in einem Programm nur ein Mal machen. Die angegebene sogenannte **Baudrate** von 9600 wird in der Marginalspalte erklärt.

So schickt man eine Nachrichtenzeile an den Computer. Die Nachricht selbst muss in doppelte Anführungszeichen (Shift + 2) geschrieben werden.

Println spricht man übrigens als „Print Line“, also „Drucke Zeile“, weil diese Anweisung nach der Nachricht immer eine neue Zeile beginnt. Es gibt noch die Anweisung Serial.print(“Hallo“), die keinen Zeilenumbruch macht.

7.1 Was wird das obige Programm im seriellen Monitor anzeigen?

7.2 Schreibe ein Programm, das ein Gesicht anzeigt. Vielleicht fallen dir noch andere ein? Schreibe Unterprogramme für „traurig“, „lachen“…

7.3 Im rechts dargestellten Programm steht das x in der Anweisung Serial.print(x) nicht in Anführungszeichen, damit nicht der Buchstabe x, sondern der Wert der Variable x an den seriellen Monitor gesendet wird.

Was wird dieses Programm anzeigen? Kannst du den Variablenüberlauf beobachten?

```
int x=0;
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.println(x);
  x=x+5;
}
```

Wie funktioniert eigentlich die Datenübertragung?
Vielleicht weißt du, dass die Daten in Mikrocontrollern und Computern als Bitfolgen, also als Folgen von Einsen und Nullen, also als viele HIGHs und LOWs gespeichert werden. Um Daten vom Arduino an den Computer zu senden, wird vom Arduino im Prinzip eine Leitung entsprechend den Daten wechselnd HIGH und LOW geschaltet. Das nennt man eine **seriell**e Übertragung.

Die **Baudrate** gibt an, wie viele Zeichen pro Sekunde gesendet bzw. empfangen werden. Der Wert in Serial.begin() und rechts unten im seriellen Monitor müssen übereinstimmen.

Hinweis:
Die serielle Verbindung nutzt die Pins 0 und 1 des Arduino. Diese Pins solltest du möglichst nicht für andere Aufgaben verwenden.

Speicherplatz:
Texte benötigen pro Zeichen ein Byte an Speicherplatz.

8 for-Schleife

Schreibt man Anweisungen in das Unterprogramm loop(){}, werden sie unendlich oft wiederholt. Um etwas nur exakt 20 oder 30 Mal ausführen zu lassen, gibt es die sogenannte for-Anweisung:

Die blau dargestellten Anweisungen sollen hier 30 Mal wiederholt werden. Dazu sind sie mit geschweiften Klammern zu einem Block zusammen gefasst.

```
void setup(){
  Serial.begin(9600);
  for(int n=0; n<30; n=n+1){
    Serial.println("Hallo");
    Serial.println(n);
  }
}
void loop(){}
```

Direkt vor diesen Block schreibt man die for-Anweisung. Diese sorgt für das Wiederholen des Blocks und zählt die Wiederholungen mit. Sie ist übrigens die einzige Anweisung, bei der die Parameter mit einem Strichpunkt getrennt werden. Dafür steht hinter ihr direkt der Block und kein Strichpunkt.

Der erste Parameter: Zum Mitzählen wird eine Variable deklariert und ihr Startwert festgelegt. Hier heißt sie n und hat zu Beginn einen Wert von 0.

Der zweite Parameter: Hier steht eine mathematische Aussage. Der Block wird nur wiederholt, solange diese Aussage wahr ist. n muss also kleiner als 30 sein.

Der dritte Parameter gibt an, was mit n bei jeder Wiederholung gemacht werden soll. Hier steht, dass n dann immer um 1 erhöht wird.

Lokale Variable
Die Variable, die du in der for-Schleife deklarierst, ist dem Arduino dann auch nur innerhalb der Schleife bekannt. Solche Variablen bezeichnet man als **lokale Variable**. Variablen, die mit einer int, long oder float-Anweisung gleich zu Beginn des gesamten Programms deklariert werden und daher überall gelten, heißen **globale Variablen**.

Mathematische Aussagen
Als mathematische Aussage in einer for-Schleife kann man neben N<30 auch N<=30 schreiben. Das bedeutet „N ist kleiner oder gleich 30“. Bei herunter zählenden Schleifen bieten sich auch N>30 oder N>=30 an.

8.1 Was macht das obige Beispielprogramm? Was verändert sich, wenn du im ersten Parameter n=5 schreibst? Was verändert sich, wenn du dann als zweiten Parameter n<50 verwendest? Was geschieht, wenn der dritte Parameter n=n+3 ist?

8.2 Was würde diese Schleife machen?

```
for(int f=100; f<1000; f=f+3)
{
  tone(11,f);
  delay(60);
}
```

8.3 Seit 2016 hat auch die deutsche Polizei eine sog. Yelp-Sirene mit einem auf– und abschwellenden Heulton. Programmiere so eine Sirene. Dazu brauchst du zwei for-Schleifen.

8.4 Lauflicht: Schließe fünf LEDs an die Pins 9 bis 13 an und programmiere ein Lauflicht: Erst blitzt die LED an Pin 9, dann die an 10, dann 11 und so weiter.

LEDs **dimmen** und **Farben**

Bisher kannst du OUTPUT-Pins HIGH oder LOW setzen und damit z.B. LEDs ein oder ausschalten. An den Pins 3, 5,6, 9, 10 und 11 kann der Arduino ein LED aber auch dimmen:

9.1 Schließe eine LED mit entsprechendem Schutzwiderstand an einen der PINs 3, 9, 10 oder 11 des Mikrocontrollers an und probiere das Verändern der Helligkeit aus:

```
void setup(){
  pinMode(3,OUTPUT);
}
void loop(){
  digitalWrite(3,HIGH);
  delay(1000);
  digitalWrite(3,LOW);
  delay(1000);
  analogWrite(3,127);
  delay(1000);
}
```

Diese Zeilen machen das, was du schon kennst: high– und low-schalten von Pin 3.

Diese Zeile schaltet Pin 3 mit einer Stärke von 127 an. Die 127 steht für $\frac{127}{255}$, also ungefähr „halb an“. Bei 255 wäre die LED komplett hell, bei 0 wäre sie komplett dunkel.

9.2 Schreibe ein Programm, das eine LED immer langsam heller und dann wieder dunkler werden lässt. Zwei for-Schleifen helfen dir dabei.

9.3 Setzt man in analogWrite() die Werte 127 oder 128 ein, bekommt die LED ziemlich genau die halbe Energie. Trotzdem kommt das unserem Auge nicht „halb hell“ vor. Finde heraus, welcher Wert wirklich „halb hell“ aussieht. Haben deine Mitschülerinnen und Mitschüler ähnliche Werte?

9.4 Rechts siehst du ein Programm, das lauter **zufällige Zahlen** auf den seriellen Monitor schreibt. Mit **randomSeed (analogRead(A1))** wird der Zufallsgenerator einmal gestartet, mit s=**random (1,7);** immer eine Zufallszahl von 1 bis 6 erzeugt und in die Variable s getan.

Kannst du das Programm so umbauen, dass du mit einer LED das zufällige Flackern einer Kerze simulieren kannst?

```
int s;
void setup() {
 randomSeed(analogRead(A1));
 Serial.begin(9600);
}

void loop {
  s=random(1,7);
  Serial.println(s);
  delay(1000);
}
```

Analog ist das Gegenteil von digital. Es bedeutet, dass Signale stufenlos oder zumindest fein gestuft verarbeitet werden.

Pulsweitenmodulation: Die Anweisung analogWrite schaltet einen Pin übrigens nicht wirklich halb stark HIGH, sondern verwendet eine Technik namens Pulsweiten-modulation (**PWM**): Der Pin wird alle 2 Millisekunden kurz HIGH und dann wieder LOW geschaltet - schneller, als das Auge es erkennen kann. Innerhalb jedes 2ms-Intervalls gibt es 255 Schritte. Je mehr dieser Schritte der Pin HIGH ist, desto heller erscheint die LED.

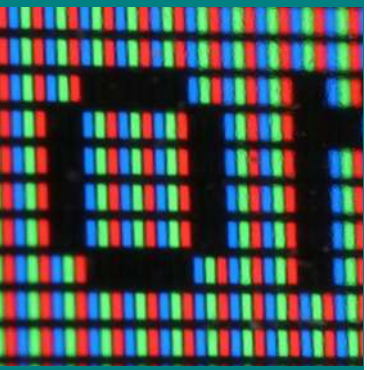
Hinweis: Die Anweisung analogWrite() funktioniert ideal mit den Pins 9 und10, bei 3 und 11 beeinflusst sie die tone()-Anweisung, bei Pins 5 und 6 geht eine LED nie ganz aus. Diese Pins sind auf vielen Arduinos mit dem Symbol ~ markiert.

Wie auch tone() schaltet analogWrite() einen Pin automatisch auf OUTPUT.

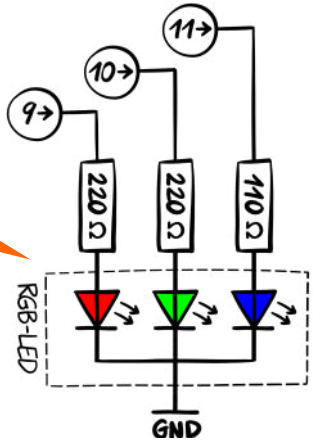
Warum rot, grün und blau?

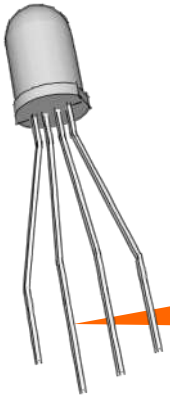
In unserem Auge werden die Farben mit Hilfe der Zapfen auf der Netzhaut wahrgenommen. Dabei gibt es drei Zapfentypen: Einen hauptsächlich für rot, einen für grün und einen für blau. Mit den drei Farben einer RGB-LED kann man die drei Zapfentypen gezielt anregen und so im menschlichen Auge jeden Farbeindruck hervorrufen.

Das nutzen auch Handydisplays, Monitore und Fernseher. Dort besteht - wenn man ganz genau hinsieht - jeder Bildpunkt aus Lichtquellen in genau diesen drei Farben:



Eine sogenannte **RGB-LED** beinhaltet drei Leuchtdioden in den Farben rot, grün und blau (daher RGB) in einem Gehäuse mit gemeinsamem Minuspol. Die Schaltung rechts zeigt dir, wie eine solche Leuchtdiode an geschlossen wird.





Alles innerhalb der gestrichelten Linie befindet sich innerhalb der RGB-LED. Du siehst also, dass die Minuspole aller LEDs zu einem gemeinsamen Minuspol (**common cathode**) zusammen gefasst sind.

Bei RGB-LEDs ist meistens das längste Anschlussbein der gemeinsame Minuspol. Welcher der anderen Anschlüsse für welche Farbe zuständig ist, probierst du am besten (mit einem 220 Ω Schutzwiderstand) aus.

9.5 Finde heraus, welcher der anderen Anschlüsse für welche Farbe ist und schließe die LED entsprechend dem obigen Schaltbild an.

Welche Mischfarben ergeben sich, wenn rot und grün, rot und blau, grün und blau sowie alle drei LEDs an sind?

9.7 Noch mehr Farbkombinationen als in Aufgabe 9.5 ergeben sich, wenn man die Farben mit analogWrite() gedimmt mischt. Schreibe ein Programm, das alle überhaupt möglichen Farbkombinationen anzeigt, also 256*256*256=16777216 Farben.

Dazu brauchst du wahrscheinlich drei for-Schleifen ineinander.

9.6 Suche die etwa zehn Tippfehler in diesem Programm.... Was soll das Programm machen?

```
void setup(); {
 serialBegin(6900);
}

void loop {
 for(int i=0;i<10;i=i+1){
  Serial.pint("Huhu");
  for(u=0,u<100;u=u+1){
    Serial.print(i);
    Serial.print(",");
    Serial.print(u);
  }
 }
```

If heißt falls

If ist eine der wichtigsten Strukturen in den meisten Programmiersprachen. Schreibt man ein if() vor eine Anweisung, kann man dafür sorgen, dass sie nur unter eine bestimmten Bedingung ausgeführt wird. Das folgende „Kopf“ oder „Zahl“-Programm zeigt dir, wie das geht:

```
int x=0;
void setup(){
 Serial.begin(9600);
 randomSeed(analogRead(A1));
 x=random(1,7);
 if(x<6) Serial.println(x);
 if(x>5) {
  Serial.print("Du hast eine 6!");
  Serial.print("Nochmal würfeln!");
 }
}
void loop(){}
```

Die **if(...)-Anweisung** funktioniert so: Nur wenn die mathematische Aussage, die zwischen den runden Klammern steht, wahr ist, wird die folgende Anweisung ausgeführt.

Damit diese beiden Anweisungen nur ausgeführt werden, wenn x größer als 5 ist, werden sie mit den geschweiften Klammern zu einem Block zusammengefasst.

Neben **x>50**, **x<50**, **x<=50** und **x>=50** kann man noch **x!=50** (ungleich) und **x==50** (gleich) schreiben. Achtung: Das „ist gleich“ schreibt man also (anders als im Mathematikunterricht) mit doppeltem Gleichheitszeichen. Natürlich sind damit auch Gleichungen möglich, z.B. x+10==y*2.

10.1 Schreibe ein Schere-Stein-Papier-Programm. Es soll nach jedem Druck auf den Reset-Knopf zufällig eines der drei Worte „Schere“, „Stein“ oder „Papier“ auf dem seriellen Monitor anzeigen. Eine Möglichkeit wäre, eine Zufallszahl zu erzeugen. Wenn sie 1 ist, wird „Schere“ angezeigt, bei einer 2 „Stein“, bei einer 3 „Papier“.

10.2 Hinter if kann man noch ein **else** schreiben. Der auf else folgende Block wird nur ausgeführt, wenn der mathematische Ausdruck in if nicht wahr war. Was wird also dieses Programm mit einer LED an Pin 9 tun?

```
long i=1;
void setup(){}
void loop(){
  if(i<100000){
    analogWrite(9,100);
  } else {
    analogWrite(9,255);
  }
  i=i+1;
}
```

Warum das doppelte Gleichheitszeichen?

In den runden Klammern der if-Anweisung muss man für die Mathematische Aussage „x ist gleich 50“ beim Programmieren x==50 schreiben. Würde man nur das einfache Gleichheitszeichen „x=50“ schreiben, bedeutet das, dass der Wert 50 in die Variable x gespeichert wird.

Schreibweise

Werden hinter if() mehrere Anweisungen in einen Block zusammen gefasst, so rückt man diese normalerweise einige Leerzeichen weit ein. Du kannst das durch einen Druck auf Strg+t aber auch automatisch machen lassen.

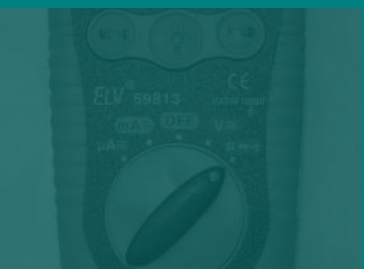
11 Eingaben mit Tastern

Schalter und Taster?

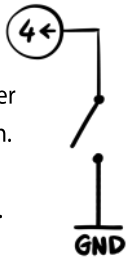
Der Unterschied zwischen Schaltern und Tastern ist: Bei Schaltern bleibt der elektrische Zustand nach dem Drücken beibehalten, bei Tastern nur, solange man drückt. Beide, Taster und Schalter, nennt man zusammen Schaltelemente.

Schaltelemente mit vielen Anschlüssen?

Viele Schaltelemente haben mehr als zwei Anschlüsse. Um sie an einem Mikrocontroller betreiben zu können, musst du heraus finden, welche der Anschlüsse beim Schalter in betätigtem Zustand verbunden und in geöffneten Zustand nicht verbunden sind. Dabei kann dir ein Multimeter als Leitfähigkeitsprüfer helfen.



Hier lernst du, wie man an den Arduino Schalter oder Taster anschließt und im Programm darauf reagiert, wenn sie gedrückt werden. Damit das funktioniert, muss ein Schalter immer zwischen einen Pin und den Minuspol angeschlossen werden. Ferner muss der Pin nicht auf den Modus OUTPUT sondern auf INPUT_PULLUP gestellt werden.



11.1 Schließe einen Taster (oder Schalter) so an Pin 4 an, wie es das Schaltbild zeigt. Dieses Programm zeigt den Zustand des Tasters im seriellen Monitor an:

```
int i;
void setup(){
  pinMode(4,INPUT_PULLUP);
  Serial.begin(9600);
}
void loop(){
  i=digitalRead(4);
  Serial.println(i);
}
```

Diese Anweisung setzt den Pin in den Modus INPUT_PULLUP.

Hier wird der momentane Zustand des Tasters abgefragt und in der Variable i gespeichert...

...und hier dann angezeigt.

Stimmt der folgende Satz? „Ist der Taster geschlossen, wird eine 0 angezeigt, weil Pin 4 dann mit dem Minuspol verbunden ist. Ist Pin 4 nicht mit dem Minuspol verbunden, weil der Taster nicht gedrückt oder überhaupt nicht angeschlossen ist, ist das Ergebnis eine 1."

11.2 Schreibe ein Programm, das eine LED an Pin 9 ausschaltet, wenn der Taster nicht betätigt ist und einschaltet, wenn er betätigt ist. If-else hilft dabei.

11.3 Schließe zwei Taster und einen Lautsprecher an. Bei Druck auf den einen soll ein tiefer Ton gespielt werden, beim anderen ein hoher Ton.

11.4 Schreibe ein Programm, das mitzählt und anzeigt, wie oft ein Taster gedrückt wird. Was geschieht beim Dauerdrücken des Tasters?

11.5 Die Anweisung **millis()** beinhaltet immer die Anzahl der Millisekunden, die der Mikrocontroller schon läuft. Kannst du mit diesem Programm herausfinden, wie lange es etwa dauert, ein Zeichen auf den seriellen Monitor zu schreiben?

```
long m;
void setup(){
  Serial.begin(9600);
}
void loop(){
  m=millis();
  Serial.println(m);
}
```

Wiederholungen mit **while**

Die while-Anweisung funktioniert ähnlich wie die if-Anweisung. Allerdings geht es bei while nicht nur darum, ob eine Anweisung oder ein ganzer Block ausgeführt wird, sondern auch, ob sie wiederholt wird:

```
while(i<100) {
  tone(10,i);
  i=i+1;
}
```

Nur wenn der mathematische Ausdruck wahr ist, wird der Anweisungsblock ausgeführt und dann solange wiederholt, bis der Ausdruck nicht mehr wahr ist.

12.1 Wie wird dieses Programm ablaufen? Kommt das Programm jemals bis zum Unterprogramm loop()?

```
int x=1;
void setup(){
 while(x<100){
  Serial.println(x);
  x=x+1;
 }
 while(2<5){
  Serial.print("!");
 }
}
void loop(){}
```

12.2 Hier ist ein Taster an Pin 4 angeschlossen. Was bewirken die beiden leeren While-Schleifen? Was macht das Programm?

```
void setup(){
 Serial.begin(9600);
}
void loop(){
 while(digitalRead(4)==1){
 }
 Serial.print("von jetzt");
 while(digitalRead(4)==0){
 }
 Serial.print("bis jetzt");
}
```

Übrigens kann man im mathematischen Ausdruck von if() und while() auch mehrere **Bedingungen** mit einander verknüpfen: if(w<100 **&&** u==3) bedeutet, dass der if-Block nur ausgeführt wird, wenn w kleiner als 100 **und** auch u gleich 3 ist. Die beiden &-Zeichen müssen ohne Leerzeichen geschrieben werden, ebenso wie hier die beiden „Pipes“, die für ein „**oder**“ stehen: if(p<10 || p>30)…. Der Block wird ausgeführt, wenn p kleiner als 10 oder p größer als 30 ist. Falsch wäre: if(p<10 || >30).

12.3 Finetuning: Für diese Aufgabe benötigst du zwei Taster und einen Lautsprecher. Immer, wenn man den einen Taster betätigt, soll der Ton um 1 Hz höher werden. Immer wenn man den anderen Taster betätigt, soll er um 1 Hz niedriger werden.

Wie kannst du zusätzlich mit einer while-Schleife verhindern, dass man pro Tastendruck mehr als ein Hertz an Frequenzänderung auslöst?

Häufiger Fehler

Es ist ein häufiger Fehler, unbeabsichtigt eine while()-Schleife so zu programmieren, dass der mathematische Ausdruck nie unwahr wird. Dann hängt das Programm für immer in dieser Schleife fest - man nennt das eine **Endlosschleife**.

13 Programme planen

Hinweis: gute Flussdiagramme

Programmablaufpläne sind Flussdiagramme, denen man sofort ansieht, wo man eine Schleife und wo eine Verzweigung programmieren muss. Sie verzichten darauf, Pfeile zu zeichnen, die quer durcheinander gehen.

Unterprogramme

Wenn man sich selbst Unterprogramme angelegt hat, zeichnet man oft für jedes Unterprogramm ein eigenes Struktogramm. Das macht es übersichtlicher!

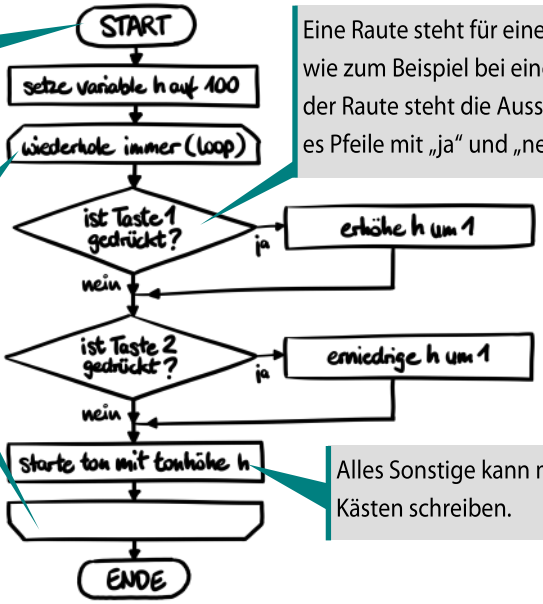
Hinweis

Mit speziellen Programmen (wie z.B. dem für private Zwecke kostenlosen PapDesigner) kannst du Programmablaufpläne auch am Computer zeichnen.

Bei größeren Vorhaben ist es sinnvoll, sich den Ablauf eines Programms zu überlegen, bevor man es programmiert. Dazu verwendet man sogenannte Programmablaufpläne (PAP), weil sie einige wenige sehr nützliche Symbole haben, aber ansonsten einfach aus freiem Text bestehen:

Los geht es immer mit einem Oval oder Kreis.

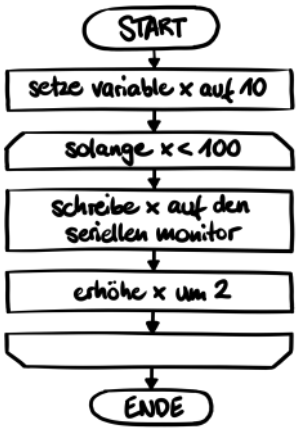
Anfang und Ende einer Schleife werden durch diese Formen dargestellt. Dazwischen steht alles, was wiederholt wird.



Eine Raute steht für eine Entscheidung, wie zum Beispiel bei einem if(…). In der Raute steht die Aussage, dann gibt es Pfeile mit „ja“ und „nein“.

Alles Sonstige kann man einfach in Kästen schreiben.

13.1 Schreibe dieses Programm:

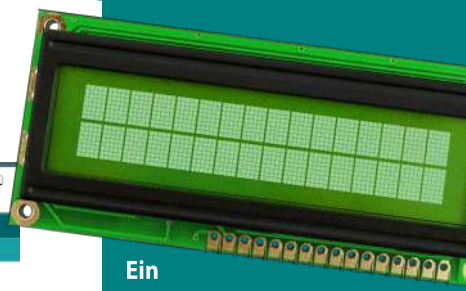


13.2 Entwerfe einen Programmablaufplan zu folgendem Ablauf: Immer, wenn man einen an Pin 4 angeschlossenen Taster betätigt, wird der aktuelle millis()-Wert auf den seriellen Monitor geschrieben.

13.3 Skizziere den Programmablaufplan zu Aufgabe 12.2 und zu Aufgabe 9.7. Wenn man mehrere Schleifen in einander zeichnet, erhöht es die Übersichtlichkeit, die äußeren etwas breiter zu zeichnen.

LCD-Display und Bibliotheken

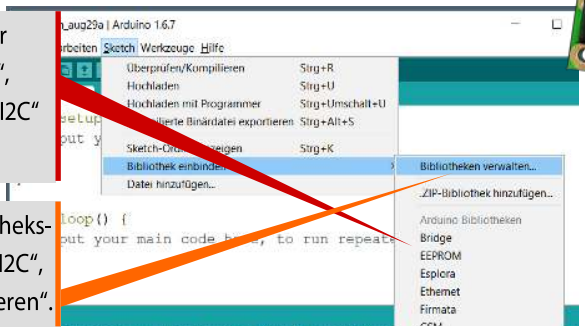
Hier lernst du, wie du in 3 Schritten ein alphanumerisches Display am Arduino betreiben kannst. Dazu bringt die Arduino-Entwicklungsumgebung die notwendigen Anweisungen noch nicht mit. Deshalb lernst du in Schritt 1, wir man eine Anweisungs-**Bibliothek** nachlädt:



Ein alphanumerisches Display ist ein LCD-Display, das hauptsächlich Buchstaben und Zahlen anzeigen kann. Das hier abgebildete Display hat 2 Zeilen für jeweils 16 Zeichen.

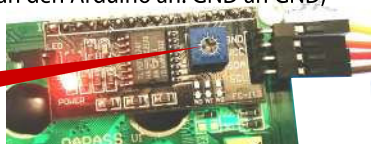
14.1 **Schritt 1:** Prüfe in der Liste unter „Sketch ⇨ Bibliothek einbinden“, ob die Bibliothek „LiquidCrystal I2C“ schon in der Liste ist.

Wenn nicht: Suche in der Bibliotheksverwaltung nach „LiquidCrystal I2C“, wähle sie aus, klicke auf „Installieren“.



14.2 **Schritt 2:** Schließe die Aufsatzplatine mit vier Kabeln an den Arduino an: GND an GND, VCC an 5 V, SDA an die SDA–Buchse und SCL an SCL.

Wenn auf dem Display nichts zu erkennen ist, kannst du hier den Kontrast verstellen.



14.3 **Schritt 3:** Probiere dieses Beispiel aus: Was macht dieses Programm?

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C Lcd(0x27,16,2);
void setup() {
  Lcd.init();
}
void loop() {
  Lcd.clear();
  Lcd.setCursor(3,0);
  Lcd.print("Zeit:");
  Lcd.print(millis());
  delay(100);
}
```

Die ersten beiden Zeilen binden zwei Bibliotheken in das Programm ein.

Diese Anweisung definiert das Display. Die 3 Parameter werden dir in der Marginalspalte erklärt.

Hier wird das Display gestartet….

… und hier wird die Anzeige gelöscht.

Hier wird der Cursor in Spalte 3 und Zeile 0 gesetzt. Links oben entspricht Spalte 0 und Zeile 0.

Die Anführungszeichen wirken wie bei Serial.print();

Probiere: Was machen die beiden Anweisungen **Lcd.backlight()** und **Lcd.noBacklight()**?

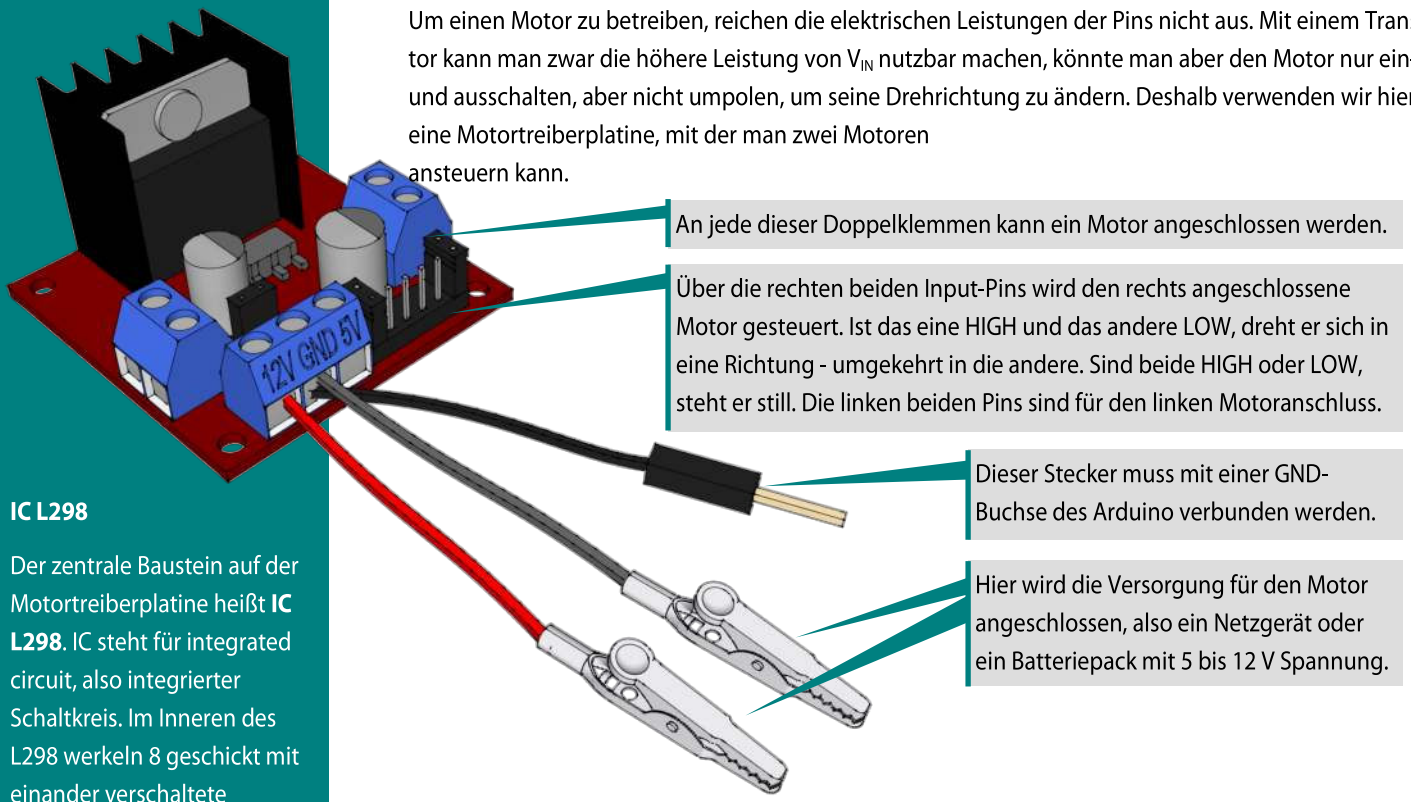
Welche Parameter?
Drei Parameter werden zur Definition des Displays benötigt: Adresse, Spaltenzahl und Zeilenzahl.

Die Adresse hängt von der rückseitig aufgesetzten I²C-Platine ab. Zu den abgebildeten Platinen passt die Adresse 0x27, es gibt aber auch 0x3f oder 0x20.

Für Displays mit 16 Spalten folgen dann die Zahlen 16, 2. Für größere Displays mit 20 Spalten und 4 Zeilen logischerweise die 20, 4.

15 Motoren steuern

Um einen Motor zu betreiben, reichen die elektrischen Leistungen der Pins nicht aus. Mit einem Transistor kann man zwar die höhere Leistung von V_{IN} nutzbar machen, könnte man aber den Motor nur ein– und ausschalten, aber nicht umpolen, um seine Drehrichtung zu ändern. Deshalb verwenden wir hier eine Motortreiberplatine, mit der man zwei Motoren ansteuern kann.



IC L298

Der zentrale Baustein auf der Motortreiberplatine heißt **IC L298**. IC steht für integrated circuit, also integrierter Schaltkreis. Im Inneren des L298 werkeln 8 geschickt mit einander verschaltete leistungsfähige Transistoren, eingebaut in ein einziges Gehäuse. Die Schaltung nennt man auch **H-Brücke**.

15.1 Schließe an die rechte Doppelklemme einen Motor an und verbinde die rechten beiden Pins mit den Pins 3 und 4 deines Arduino. Schreibe nun ein Programm, das immer wieder Pin 3 HIGH und Pin 4 LOW schaltet, zwei Sekunden wartet, dann beide Pins HIGH schaltet und wieder zwei Sekunden wartet.

Wenn du jetzt ein Netzteil bzw. Batteriepack anschließt, sollte der Motor laufen, stehen, laufen, stehen....

15.2 Schließe nun zwei Motoren an und schreibe Unterprogramme für „beide vorwärts“, „beide rückwärts“, „links vorwärts“, „rechts vorwärts“ und „stopp“.

Mit solche Unterprogrammen bist du der Steuerung eines Fahrzeugs schon recht nahe, das zwei Motoren als Hinterräder benutzt: