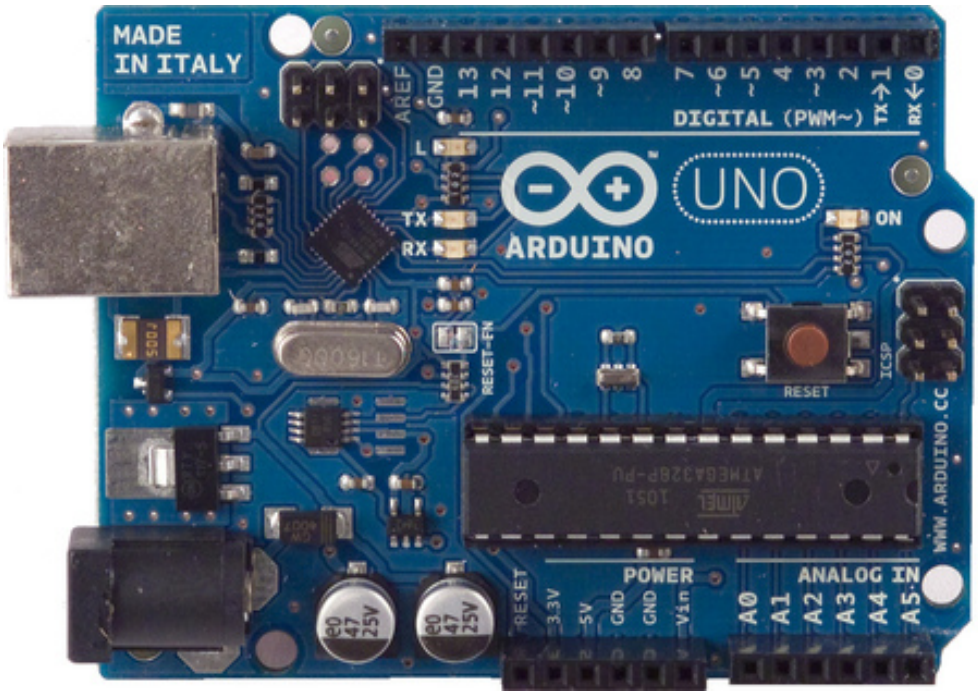


Steuerung von Geräten mit dem Mikrocontroller

von

Annemarie Bauer
Dr. Birgit Berger
Dr. Daniel Roth
Katrin Fischer
Michael Schnaus

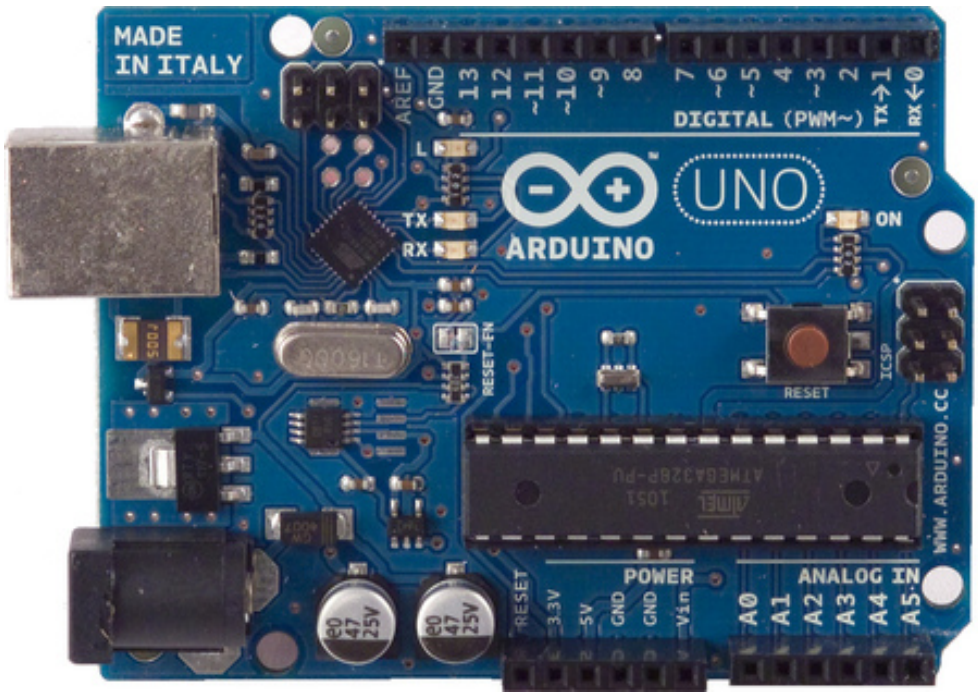
Lessing-Gymnasium
Karlsruhe

Inhalt

1. Der Arduino Mikrocontroller

1.1 Die Hardware

Das Arduino UNO - Board, das wir verwenden, ist mit einem ATmega328-Mikrocontroller ausgestattet, der quasi das Herzstück des Boards bildet. Dennoch sind einige weitere Komponenten wichtig und nötig, um mit dem Arduino arbeiten zu können.



➔ A 1.1 **Lies den folgenden Abschnitt aufmerksam durch und beschrifte die Abbildung auf dem Arbeitsblatt "Arduino UNO - Die Hardware".**

- **USB-Port:** Damit lässt sich der Arduino leicht an einen PC anschließen und kann mit diesem kommunizieren. Programme, die am PC geschrieben wurden, können direkt auf den Arduino übertragen werden. Der PC übernimmt dann auch die Energieversorgung des Arduinos.
- **Externe Spannungsversorgung:** Ist der USB-Port nicht verbunden, muss die Spannungsversorgung extern erfolgen - beispielsweise durch das Anschließen einer 12V-Batterie. Es wird empfohlen, Spannungen zwischen 7V und 12V zu verwenden. Andernfalls kann das Board beschädigt werden.
- **Digitale Pins:** An den 14 digitalen Pins können Bauteile wie LEDs mit dem Board verbunden werden. Je nach Programmierung kann jeder der Pins Input oder Output sein. Digitale Pins können nur mit den logischen Zuständen 0 oder 1 arbeiten.
- **Analoge Pins:** Das Arduino-Board verfügt über 6 analoge Ein- beziehungsweise Ausgänge, die mit A0 bis A5 gekennzeichnet sind. Im Gegensatz zu digitalen Sensoren, die entweder HIGH (1) oder LOW (0) sind, können analoge Sensoren auch Zwischenwerte verarbeiten. Analoge Sensoren sind beispielsweise Temperaturfühler oder Lichtsensoren.

- **Power Pins:** Das Arduino Board regelt die extern angelegte Spannung selbstständig auf den aufgedruckten Wert. Der GND-Pin ist die Erdung.
- **Mikrocontroller:** Das wichtigste Bauteil auf dem Arduino-Board ist natürlich der Mikrocontroller des Typs ATmega328. Er besitzt unter anderem einen eigenen Speicher von 32 KB.

1.2 Die Entwicklungsumgebung

Was ist überhaupt eine Entwicklungsumgebung?
Dabei handelt es sich um die Software zu unserem Mikrocontroller, die wir am Laptop oder iPad öffnen können und welche das Programmieren des Mikrocontrollers ermöglicht. Kurz gesagt, wird das eigentliche Programm in der passenden Programmiersprache (hier C und C++) in dieser Entwicklungsumgebung geschrieben und anschließend auf den Arduino übertragen.
Eine solche integrierte Entwicklungsumgebung wird kurz IDE (**I**ntegrated **D**evelopment **E**nvironment) genannt.

Die Arduino IDE ist eine "freie Software", die du dir auch zu Hause kostenlos herunterladen kannst. Sie hat dabei einige wichtige Funktionen, die das Programmieren erleichtern:

- **Farbliche Hervorhebung des Codes:** Um die Übersicht innerhalb des Programms zu behalten, werden verschieden Befehle automatisch in verschiedenen Farben dargestellt. Da du noch nicht weißt, wie so ein Code aussieht, kannst du dir noch nicht viel darunter vorstellen, deshalb später mehr dazu.
- **Überprüfung des Codes:** Wie bei einem Aufsatz in Deutsch, kann es vorkommen, dass sich beim Schreiben eines Programms Fehler einschleichen, wodurch der Code keinen Sinn mehr ergibt. Manchmal ist es nur ein einziges fehlendes Zeichen und schon funktioniert gar nichts mehr. Daher verfügt die IDE über die Funktion "Überprüfen". Der Code wird kontrolliert und das Programm zeigt an, ob und wo sich ein Fehler befindet.
- **Upload:** Sind alle Fehler beseitigt, kann in der IDE der Befehl zum Upload gegeben werden. Das Programm wird dann zum Mikrocontroller geschickt.

➔ A 1.2.1: **Öffne jetzt die Arduino IDE und mache dich mit den Funktionen vertraut. Du findest das Programm im von deinem Lehrer genannten Ordner.**

1.3 Einführung in die Programmstruktur

1.3.1 setup-Block und loop-Block

Jedes Programm besteht aus mindestens zwei Blöcken, dem setup-Block und dem loop-Block:

```
void setup(){
      anweisungen;
}

void loop(){
      anweisungen;
}
```

Die Anweisung, die im `setup` - Block steht, wird nur einmal ausgeführt, was im `loop`-Block kommt, wird in einer Endlosschleife immer wieder ausgeführt. Im `setup`-Block können auch Anweisungen stehen, die das eigentliche Programm vorbereiten. Die Blöcke müssen immer vorhanden sein, auch wenn keine Anweisung folgt. Das eigentliche Programm steht meist im `loop`-Block.

Zu beachten ist, dass alle Anweisungen in geschweiften Klammern stehen. Jede offene geschweifte Klammer { muss immer am Ende des Blocks wieder geschlossen werden }, sonst gibt es eine Fehlermeldung. Diese gibt es auch, wenn man nicht jede Zeile mit einem Semikolon (;) beendet.

Achte auch auf die Groß- und Kleinschreibung.

Möchte man sich eine Notiz in eine Zeile eintragen - beispielsweise, damit man später nachvollziehen kann, was der Befehl im Programm bewirkt – dann kann man diese Notiz durch // von den Befehlen abtrennen. Alles was nach // steht fließt nicht in das Programm ein und braucht daher keinen Speicherplatz.

Längere Notizen über mehrere Zeilen hinweg beginnt man mit /* und beendet man mit */.

Solche Notizen nennt man „Kommentare“ bzw. das Erstellen solcher Notizen „kommentieren“.

1.3.2 "Hallo Welt"

Nachdem du die Arduino IDE bereits im Abschnitt 1.2 kennen gelernt hast, kann es schon mit dem ersten Programm (auch Sketch genannt) losgehen. Unser erstes Programm nennt sich “Hallo Welt” und wird gerne verwendet, um eine neue Programmiersprache einzuweihen. Als erstes öffnest du die Arduino IDE, nebenstehender Bildschirm sollte euch erwarten.

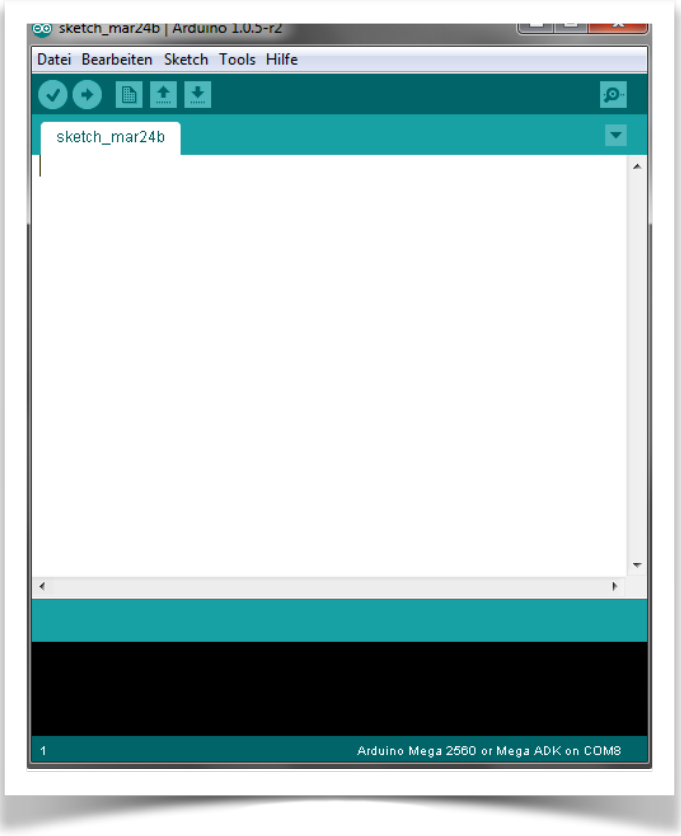
Anschließend wählst du den Menüpunkt **Werkzeuge ➔ Board** aus und suchst in der Auswahlliste unseren Arduino-Typ, nämlich den Arduino Uno. Somit weiß das Programm mit welchen Arduino es zu tun hat.

Verbinde den Arduino Uno mit dem Laptop. Warte, bis der Laptop den Arduino erkannt hat, es sollte dann mindestens ein Lämpchen am Arduino leuchten. Wähle außerdem im Menü **Werkzeug ➔ Port** den **Com-Anschluss** aus, hinter dem **Arduino UNO** steht!

Schreibe nun dein erstes Programm. Du brauchst dafür folgende Befehle:

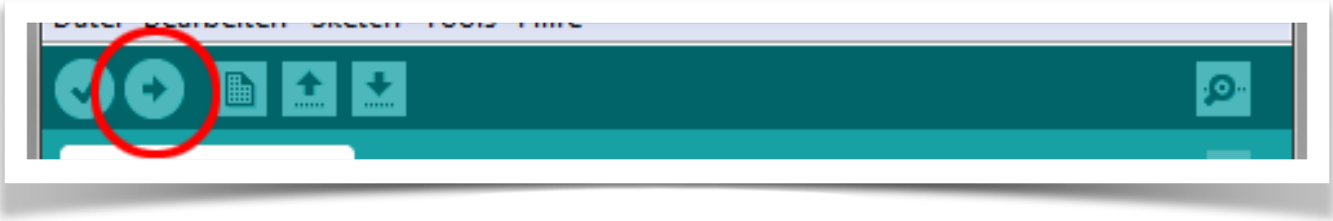
```
Serial.begin(9600); /* steht im setup-Block und richtet eine serielle Schnittstelle ein, die du brauchst, um einen Text anzeigen zu lassen */
```

```
Serial.println("DEIN TEXT");
```
oder
```
Serial.print("DEIN TEXT"); /  * gibt deinen in Klammern und Anführungs-zeichen eingegebenen Text aus */
```



➔ A1.3.2 a) **Schreibe ein Programm, das unendlich oft „Hallo Welt“ ausgibt.**

Um nun zu prüfen, ob der Code richtig eingeben wurde und gelesen werden kann, klicke auf den Haken oben links, der die Überprüfung startet. Ist diese abgeschlossen, erhälst du die Information „Kompilierung abgeschlossen“.



Nun kann der Code (in der Arduino-Sprache auch Sketch genannt) übertragen werden. Drücke dazu den Pfeil oben links.

Wenn die Übertragung erfolgreich war, erscheint im unteren Bereich des IDE Bildschirms “Upload abgeschlossen”. Sollte es ein Problem geben, erhältst du in diesem Feld eine Fehlermeldung der Form: `not in sync resp=0x00`. Dann solltest du überprüfen, ob der Arduino korrekt angeschlossen ist und alle Einstellungen (siehe oben!) vorgenommen sind. Eventuell hilft es, den Arduino erneut anzuschließen oder die USB-Schnittstelle zu wechseln.

Da der Arduino selbst keinen Ausgabebildschirm hat, lassen wir uns die Ausgabe (Hallo Welt) am Laptop anzeigen, dazu wählen wir folgendes Symbol in der IDE aus:

Nun öffnet sich ein neues Fenster, in der die Ausgabe des Arduinos dargestellt wird.

Hat es geklappt? Herzlichen Glückwunsch! Du hast dein erstes Programm geschrieben!

Zeige es bitte deinem Lehrer und speichere es auf deinem USB-Stick unter einem **sinnvollen Dateinamen** ab! Lege gegebenenfalls einen Ordner „NWT“ an.

➔ A1.3.2 b) **Schreibe das Programm so um, dass es nicht mehr „Hallo Welt“, sondern „Hallo *dein Name*“ ausgibt.**

➔ A1.3.2 c) **Schreibe das Programm so um, dass der Arduino nur noch einmal "Hallo Welt" sagt.**

➔ A1.3.2 d) **Kommentiere das Programm.**

➔ A1.3.2 e) **Worin liegt der Unterschied zwischen den Befehlen** `Serial.println` **und** `Serial.print` **? Untersuche dies mithilfe des folgenden Programms:**

```
void setup(){
      Serial.begin(9600);
      Serial.print("Text1");
      Serial.println("Text2");
      Serial.print("Text3");
void loop(){}
```

➔ A1.3.2 f) **Überlege dir weitere Ausgaben für den Arduino und programmiere diese.**

*Hinweis: Du hast nun die ersten Befehle kennen gelernt. Beginne daher **jetzt** damit, dir eine übersichtliche Tabelle mit allen Befehlen anzulegen!*

2. Die Simulation - Tinkercad Circuits

2.1 Die Simulation

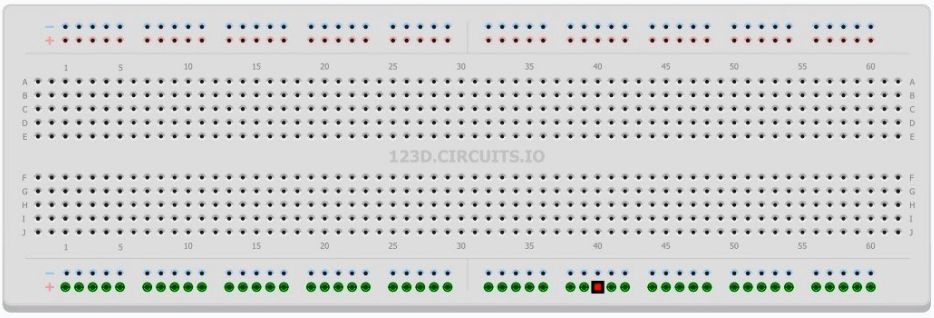
Unsere Simulationssoftware nennt sich Tinkercad Circuits. So eine Simulationssoftware ermöglicht es Schaltungen zu erproben oder neue Schaltungen zu entwickeln, ohne tatsächlich die Bauteile zu benötigen. Man weiß also schon bevor man an die tatsächliche Umsetzung geht, ob alles funktioniert (oder funktionieren sollte) und schützt damit die Bauteile vor falscher Handhabung.

Für die Anmeldung rufst du im Internet die Seite https://www.tinkercad.com/ auf, klickst auf "Anmelden" und meldest dich als „Schüler mit Klassencode“ mit den zugeteilten Zugangsdaten an.

Auf der Startseite findest du ab jetzt deine Projekte - wähle dazu Schaltkreise(engl. Circuits) aus. Klicke auf die Schaltfläche **„Neuen Schaltkreis erstellen"** um dein erstes Projekt zu beginnen.

2.2 Das Breadboard

Ein Breadboard ist ein Steckbrett, auf dem elektronische Bauteile aufgesteckt werden können. Die Steckplätze für die Beinchen der Bauteile nennen sich Pins. Diese Pins sind nach einem bestimmten Prinzip miteinander verbunden, aber dazu später mehr.



Um dich mit der Simulation und dem Breadboard vertrauter zu machen, hier ein paar **Übungen**:

1. Stecke ein Bauteil auf das Breadboard. Du kannst hierfür die Suche benutzen. Achte darauf, dass es angeschlossen ist.
2. Drehe das Bauteil.
3. Lösche das Bauteil.
4. Verbinde zwei Pins mit einem Kabel.
5. Stecke eine LED und verändere anschließend ihre Farbe.
6. Verkleinere und vergrößere die Ansicht durch Zoomen.
7. Speichere die Schaltung unter dem Namen "Erster Versuch“.

   Tipp: Das Speichern geht automatisch. Zum Umbenennen klickst du in das Feld oben links, in dem deine Schaltung bereits einen zufälligen Name erhalten hat.

Auf das Breadboard kannst du wie gesagt die Bauteile stecken, die der Mikrocontroller steuern soll. Dazu hat das Breadboard verschiedene Anschlüsse, die zum Teil miteinander leitend verknüpft sind.

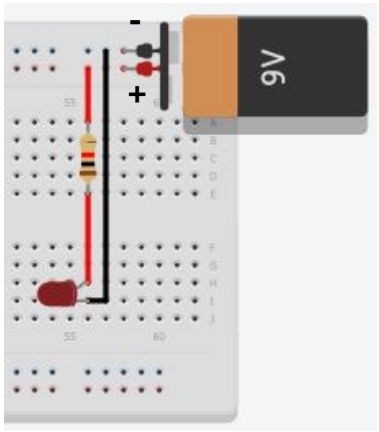
In der Simulation siehst du immer, welche Pins mit dem markierten Pin (in der Abbildung rot) verbunden sind, indem sie grün hinterlegt werden.

➔ A2.2 **Finde mit Hilfe der Simulation heraus, welche Pins jeweils miteinander leitend verbunden sind.**

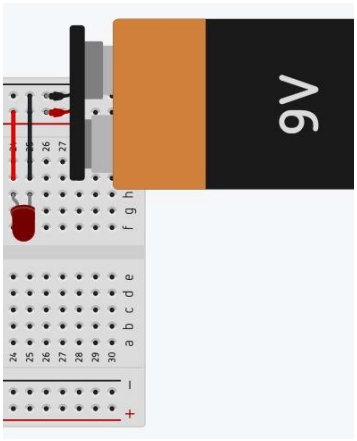
2.3 Los geht's! Erste Schritte in Simulation und Wirklichkeit

➔ A2.3 a) **Baue die drei Schaltungen in der Simulation auf. Lege dazu jeweils eine neue Kopie mit dem Duplizieren-Befehl an. Teste in welcher der drei Schaltungen die LED leuchtet. Um die Simulation zu testen, musst du rechts oben die Simulation starten.**

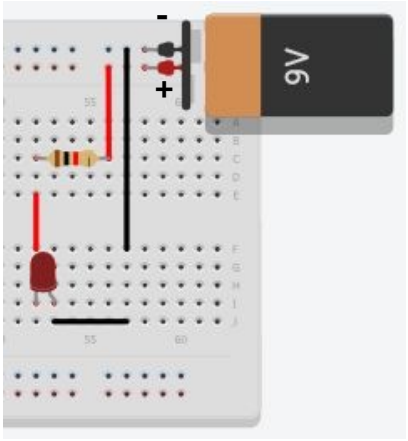
(1)



(2)



(3)



➔ A2.3 b) **Zeichne jeweils den Schaltplan und erkläre, warum die LED leuchtet oder nicht.**

➔ A2.3 c) **Warum macht es Sinn, die Batterie wie in den Abbildungen anzuschließen?**
Tipp: Beachte die Farbgebung der entsprechenden Pins auf dem Breadboard.

➔ A2.3 d) **Baue die Schaltung so auf, dass die LED leuchtet.**

Falls du Probleme hast, sind hier ein paar Tipps:

1. *Die LED hat eine Richtung in der sie den Strom leitet. In der anderen sperrt sie und leuchtet nicht.*
2. *Passen Spannungsquelle und LED zusammen (ähnliche Nennspannung)?*
3. *Eine rote LED darf nur mit 1,6 Volt betrieben werden. Wie regelst du die Spannung herunter?*
4. *Beachte die leitenden Verbindungen im Breadboard - siehe dazu A2.2!*

➔ A2.3 e) **Erstelle eine Merkregel für den Anschluss einer LED. Gehe dabei vor allem auch auf die Fehler ein, die man bezüglich Tipp 1 und 3 machen kann.**

➔ A2.3 f) **Finde heraus, was der kleinste Widerstand ist, mit dem man die grüne LED betreiben kann.**
Hinweis: Der Widerstandswert kann bei laufender Simulation geändert werden!

➔ A2.3 g) **Statt einer Batterie benutzt du im Realexperiment den Arduino als Spannungsquelle. Überlege (oder teste oder lese nach), welche der Power-Pins du wie anschließen musst.**

So und jetzt geht es in die Praxis.

➔ A2.3 h) **Wenn deine Simulation eine leuchtende LED hervorgebracht hat, kannst du nach Rücksprache mit dem Lehrer das Ganze in echt nachbauen.**

Die nächsten Schritte werden wir wieder gemeinsam machen. Wenn du hier erfolgreich angelangt bist, helfe bitte deinen Mitschülern bei der Bearbeitung und Fehlersuche.

3. Schaltungen mit LEDs

3.1 Verschiedene Blink-Schaltungen

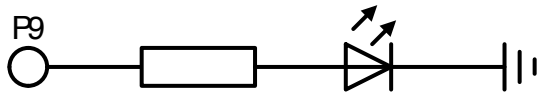
Bisher hast du gelernt, wie man eine LED dauerhaft zum Leuchten bringt. In diesem Abschnitt lernst du, wie du mithilfe eines Programm steuern kannst, wie oft und wie lange eine LED blinken soll.

Dazu muss mindestens ein Anschluss der LED zwischen hohem und niedrigem Potential wechseln können. Dies ist mit den digitalen Pins am Arduino mit den folgenden Befehlen möglich.

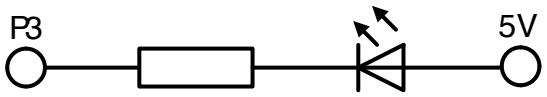
```
digitalWrite(6, HIGH); //setzt Pin 6 auf hohes Potenzial (5V)
digitalWrite(7, LOW); //setzt Pin 7 auf niedriges Potenzial (0V = GND)
```

Pins werden im Schaltplan durch einen Kreis mit entsprechender Beschriftung dargestellt.

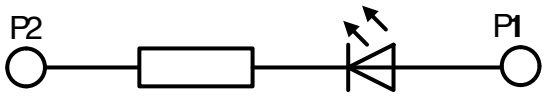
➔ A3.1.1 a) Welchen Befehl muss der Mikrocontroller bekommen, damit die LED leuchtet?



b) Welchen Befehl muss der Mikrocontroller bekommen, damit die LED leuchtet?



c) Welchen Befehl muss der Mikrocontroller bekommen, damit die LED leuchtet?



➔ A3.1.2 a) **Baue eine der drei Schaltungen aus A3.1.1 in der Simulation auf und schreibe ein Programm, das die LED zum Leuchten bringt.**

Hinweis: Immer wenn man das Potential eines Pins ansteuern möchte, muss man den Pin als Ausgang definieren (hier Pin 6). Dazu muss folgender Befehl im **Setup-Block** stehen:

```
pinMode(6, OUTPUT);
```

➔ A3.1.2 b) **Wenn die Schaltung in der Simulation funktioniert, kopiere den Code in das Arduino-Programm und teste die Schaltung mit echten Bauteilen.**

Tipp: Speichere jedes Programm auf deinem USB-Stick ab.

Ab jetzt: Verwende immer zuerst das Simulationsprogramm, um deine Schaltungen zu testen. Erst wenn die Schaltung in der Simulation funktioniert, darfst du sie in echt nachbauen und deinem Lehrer zeigen.

Unser Gehirn kann nur eine begrenzte Anzahl von Bildern pro Sekunde verarbeiten und nimmt ein zu schnelles Flimmern einer LED als ein kontinuierliches Leuchten wahr. Damit du eine LED blinken siehst, musst du also Pausen in dein Programm einbauen. Pause bedeutet, dass alles so bleibt, wie es gerade ist, eine eingeschaltete LED bleibt also an, eine ausgeschaltete LED bleibt aus. Erst nach Ablauf der Pause wird der Programmcode weiter abgearbeitet. Die Dauer der Pause kannst du selbst festlegen. Beachte dabei, dass Zeitangaben immer in der Einheit Millisekunden gemacht werden.

```
delay(1000); //1 Sekunde Pause
```

➔ A3.1.3 **Ändere das Programm aus Aufgabe A3.1.2 so ab, dass die LED immer abwechselnd eine Sekunde an und eine Sekunde aus ist.**

Hinweis: Lege dazu eine Kopie von deinem Programm an!

➔ A3.1.4 **Schreibe ein Programm, das eine LED genau dreimal für jeweils 2 Sekunden leuchten lässt.**

➔ A3.1.5 **Schließe eine zweite LED an Pin 12 an und schreibe ein Programm, das die beiden LEDs immer abwechselnd jeweils 0,2 Sekunden lang leuchten lässt.**

3.2 Anwendung: Ampel

Für die Ampelschaltung benötigst du drei LEDs (rot, gelb, grün).

➔ A3.2.1 a) **Schließe die LEDs an die folgenden Pins an:**
rot ➔ 9; gelb ➔ 8; grün ➔ 7
Verwende nur einen Widerstand für alle drei LEDs und zeichne den Schaltplan auf.

Damit du dir nicht merken musst, welcher Pin zu welcher LED gehört, ist es hilfreich, anstelle der Zahlen 7, 8 und 9 aussagekräftige Namen zu verwenden. Dies macht man mit sogenannten Konstanten, die man üblicherweise vor dem `setup`-Block des Programms folgendermaßen definiert:

```
const int rot = 9;
```

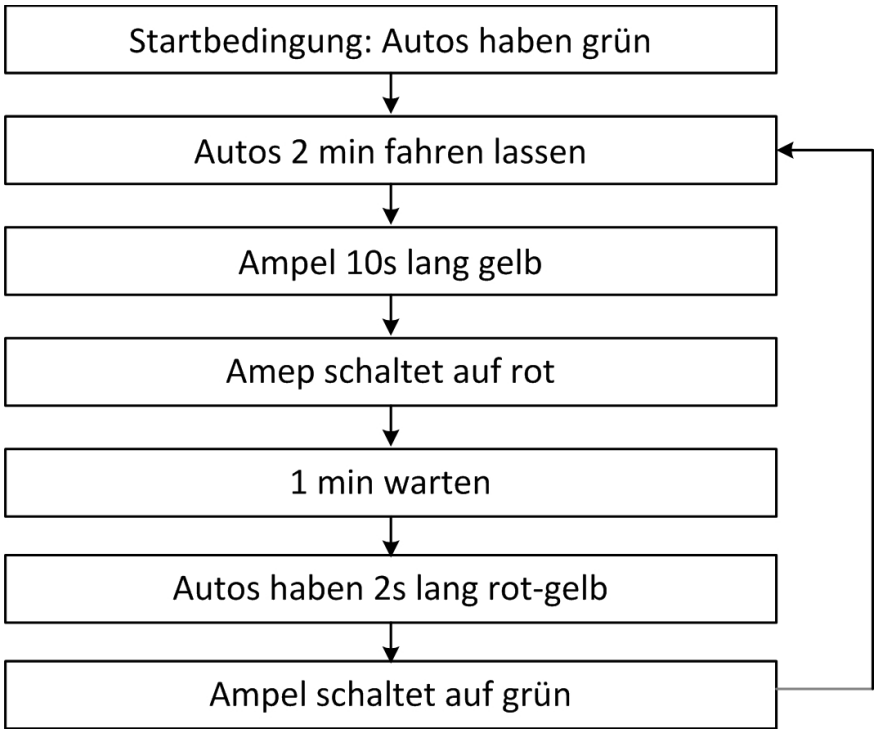
Der Konstanten `rot` ist nun der Wert 9 zugeordnet. Wenn du nun Pin 9 auf HIGH schalten möchtet, kannst du anstelle von `digitalWrite(9, HIGH);` auch `digitalWrite(rot,HIGH);` schreiben.

➔ A3.2.1 b) **Beginne mit dem Schreiben des Ampelprogramms. Überlege dir für alle drei LEDs aussagekräftige Namen (keine Umlaute!) und definiere entsprechende Konstanten vor dem `setup`-Block. Lege anschließend alle verwendeten Pins im `setup`-Block als Ausgänge fest. Das restliche Programm ist erst in Aufgabe A3.2.1c) zu schreiben.**

Bevor du das eigentliche Programm schreibst, solltest du dir klar machen, was die Startbedingungen für die Ampelschaltung sind, welche LEDs gleichzeitig und welche nacheinander

an- oder ausgeschaltet werden sollen, an welchen Stellen Pausen eingebaut werden müssen und wo die Endlosschleife beginnt, d.h. welche Programmzeilen im `setup`- und welche im `loop`-Block stehen müssen. Am besten gelingt das mit einem sogenannten Ablaufdiagramm (siehe Abbildung).

➔ A3.2.1 c) **Schreibe das Ampelprogramm mithilfe des Ablaufdiagramms fertig und kommentiere es.**



Hinweis: Möchtest du deine Ampelschaltung im Schnelldurchlauf testen, musst du alle Zeiten einzeln ändern und später wieder von Hand auf den Anfangswert zurücksetzen. Einfacher geht das wenn du im Programm Konstanten verwendest denen du am Anfang des Programms einmal ihren Wert zuweist, z.B.

```
const int zeitGelb = 10000; // 10 Sekunden Dauer
const int zeitGruen = 60000; //1 Minute Dauer
```

Verwendung der Konstanten im Programm:

```
delay(zeitRot);
delay(zeitGruen);
```

oder du Verwendest einen Faktor, mit dem du gleichzeitig alle Zeiten verkürzen oder verlängern kannst:

```
const int zeitfaktor = 1000;
delay(10*zeitfaktor); // 10 Sekunden Dauer
delay(60*zeitfaktor); //1 Minute Dauer
```

4. Variablen

Du hast bestimmt schon mal ein Computerspiel gespielt. Bei den meisten Spielen wird irgendwo der Spielstand angezeigt, z.B. in Form von Punkten oder von erarbeiteten „Geld“. Dieser Wert ist ja nicht immer der gleiche. Er muss also abgespeichert und verändert werden können und man muss, z.B. für die Ausgabe am Bildschirm, darauf zugreifen können.

Wenn du in Wirklichkeit etwas bekommst, das du behalten und wiederverwenden möchtest, legst du es in eine Kiste, die du beschriftest sodass du den Gegenstand später wieder findest.
Variablen in einem Programm kann man sich als solche Kisten vorstellen, wobei jede Kiste die drei folgenden Eigenschaften besitzt:

- Typ = Form und Größe der Kiste
- Name = Beschriftung der Kiste
- Wert = Inhalt der Kiste

4.1 Typ und Benennung

Ein einzelner Buchstabe braucht natürlich viel weniger Speicherplatz als ein ganzer Text, eine kleine ganze Zahl weniger als eine Dezimalzahl mit 12 Nachkommastellen. Deshalb reserviert man nicht für jeden Wert, den man abspeichern möchte, gleich viel Speicher.

Stell dir ein großes Containerschiff vor: Es gibt darauf eine große Menge von „Kisten“, alle haben die gleiche Größe. Darf in jeder Kiste nur höchstens ein Gegenstand sein (also z.B. ein Auto oder eine Banane), so würde ziemlich viel Platz auf dem Schiff verschwendet.

Auch im Computer beinhaltet jede Variable nur einen Wert. Um Speicherplatz effizient zu nutzen, werden also verschieden große Kisten benötigt.

Dafür gibt es verschiedene Variablentypen, z.B. byte, int, float, long und String.
int (Integer) ist der meist verwendete Datentyp für die Speicherung von ganzzahligen Werten ohne Dezimalkomma. Sein Wert reicht von -2^15=-32.768 bis 2^15-1=32.767.
long speichert ebenfalls ganze Zahlen, es wird jedoch mehr Speicherplatz reserviert, so dass die Werte von -2^31 bis 2^31-1 reichen.
float wird für das Speichern von Dezimalzahlen verwendet.

Zum Namen: Grundsätzlich kann man Variablennamen ziemlich frei aus Buchstaben und Zahlen zusammensetzen. Verboten ist dabei aber der Gebrauch von Spezialzeichen wie Leerzeichen, Umlauten, Akzenten und Satzzeichen. Damit man den eigenen (und fremden) Quellcode besser lesen kann, hält man sich zudem an folgende Regeln (= Konventionen):

- Variablen werden klein geschrieben - z.B. vorname statt Vorname.
- Besteht eine Variable aus mehreren Teilwörtern, so schreibt man bei jedem folgenden Teilwort den ersten Buchstaben groß: z.B. geburtsOrt statt geburtsort oder geburts_ort.
- Verwende selbsterklärende Variablennamen, also z.B. alterInJahren statt x.

4.2 Variablen deklarieren und Startwert zuweisen

Um eine Variable benutzen zu können, müssen wir sie zunächst „deklarieren“. Bildlich gesprochen heißt das, dass wir die Kiste bauen und beschriften.

Die Befehlszeile `int meineZahl;` erstellt nun also eine Variable des Typs „int“ und gibt ihr den Namen „meineZahl“. Dabei steht der Typ immer vor dem Namen und das Ganze wird mit einem Semikolon (Strichpunkt) abgeschlossen.

Nun kann man dieser Variablen einen Startwert zuweisen, z.B. mit:

```
meineZahl = 2;
```

Wenn es sich anbietet, kann man die Deklaration und die erste Zuweisung einer Variable auch in einer einzigen Zeile erledigen:

```
int meineZahl = 2;
```

4.3 Rechenoperationen

Den deklarierten Variablen kann nun immer wieder ein neuer Wert zugewiesen werden, d.h. es werden neue Dinge in die Kiste gelegt und der alte Inhalt entfernt, z.B., wenn ihr Rechenoperationen ausführen möchtet.

Die Rechenzeichen sind + (Addition), - (Subtraktion), * (Multiplikation) und / (Division).

Beispiele:

```
x = 5;      //Der Variablen x wird der Wert 5 zugeordnet.

y = 3;      //Der Variablen y wird der Wert 3 zugeordnet.

z = x + y; /* Der Variablen z wird der Wert der Summe von x und y,
              d.h. der Wert 8 zugeordnet. */

x = x + 1; /* Der Wert von x erhöht sich um 1, d.h. der neue Wert
              von x ist x = 5+1 =6*/
```

Die letzte Zeile ist folgendermaßen zu lesen: neuer Wert für x = alter Wert für x + 1. Es werden also immer die alten Werte in den Rechenausdruck rechts des Gleichheitszeichens eingesetzt um den neuen Wert der Variablen zu berechnen.

Falls ich also auf der Straße einen 100€-Schein finde, so kann ich den Effekt mit folgender Zuweisung ausdrücken:

```
meinVermoegen = meinVermoegen + 100;
```

Falls ich meinen Nachbarn überfalle und mir all sein Geld aneigne dann mit:

```
meinVermoegen = meinVermoegen + nachbarsVermoegen;
nachbarsVermoegen = 0;
```

Und wenn ich dann aus Reue die Hälfte meines Geldes dem Förderverein der Schule spende, so wirkt sich dies aus:

```
meinVermoegen = meinVermoegen*0,5;
```

4.4 Variablen-Werte verwenden

Um den Wert einer Variablen zu verwenden schreibt man einfach den Namen der Variablen:

```
Serial.println(meineZahl);      // gibt am Seriellen Monitor den Wert
                                   der Variablen meineZahl aus

delay(zeitAmpel);               // die delay-Zeit hängt vom Wert der Variablen
                                   zeitAmpel ab
```

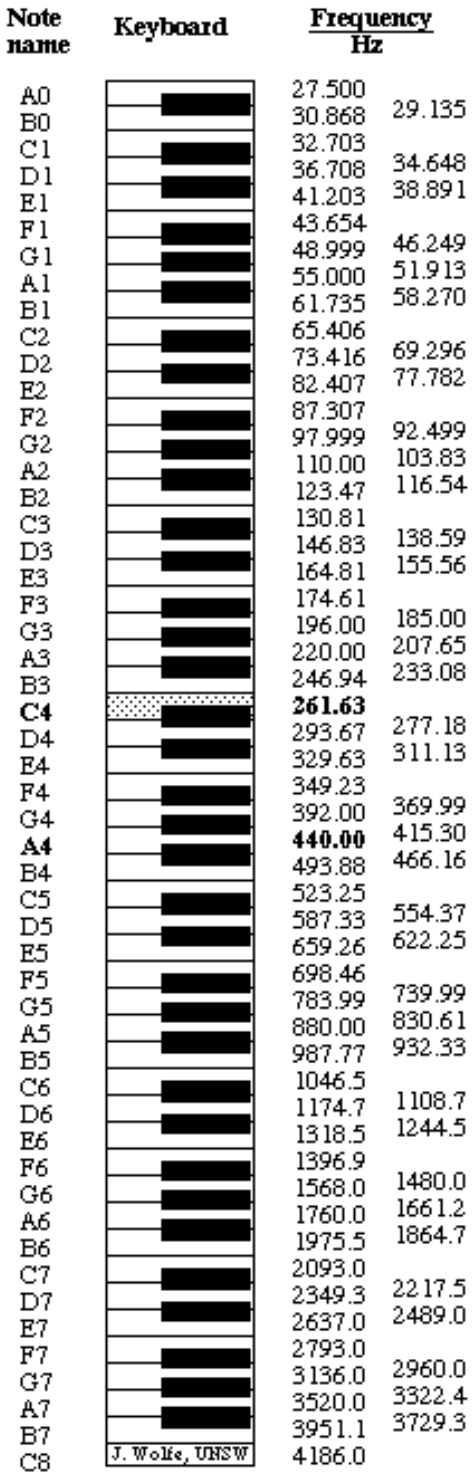
➔ A4.4.1 **Definiere eine Variable vom Typ int mit dem Namen haus und gib ihr den Wert 2.**

➔ A4.4.2 **Gib den Wert folgender Variablen am Ende der jeweiligen Aufgabe an:**

```
int anzahlBaelle = 4+3;
int anzahlTische = anzahlTische*2;
int anzahlPunkte = 0;
anzahlPunkte = anzahlPunkte + 1;
anzahlPunkte = 4;
anzahlPunkte = anzahlPunkte + 2;
```

5. Der Lautsprecher

In diesem Kapitel lernst du, wie man mit dem Mikrocontroller Töne erzeugen kann. Dazu benötigst du einen Lautsprecher. Ein Lautsprecher ist im Wesentlichen ein Elektromagnet mit einer Membran. Immer, wenn ein Strom durch den Elektromagneten fließt, zieht er die Membran nach hinten. Fließt kein Strom, schwingt die Membran wieder nach vorn. Geschieht beides abwechselnd mit einer hinreichend hohen Frequenz, so erklingt ein für den Menschen hörbarer Ton.



http://newt.phys.unsw.edu.au/jw/notes.html

Hinweis: Wie bei der LED muss das lange Bein des Lautsprechers am hohen Potential angeschlossen und je nach Bautyp ein Vorwiderstand verwendet werden.

Aus Physik in der 7. Klasse könntest du noch wissen, dass die Tonhöhe von der Frequenz abhängt, d.h. von der Anzahl der Schwingungen der Membran pro Sekunde. Wenn du also die Frequenz der Membran veränderst, kannst du die Tonhöhe verändern und damit schöne Lieder komponieren.

Dazu dient der Befehl:

```
tone(Pin, Frequenz, Dauer);
```

Frequenz: wird in Hertz (Hz) angegeben
Dauer: wird in Millisekunden (ms) angegeben

Beim Programmieren einer Melodie muss zusätzlich nach jedem Ton ein `delay` in Länge der Tondauer folgen. Alternativ kannst du den Ton mit

```
tone(Pin, Frequenz);
```

einschalten und mit Hilfe des Befehls `noTone(Pin);` die Tonerzeugung stoppen.

➔ A5 a) **Schließe den Lautsprecher an einem Pin an und schreibe ein Programm so, dass einmal für genau 2 Sekunden ein Ton abgespielt wird.**

➔ A5 b) **Lasse einen Lautsprecher im 2 Sekunden-Takt piepsen. Das heißt der Lautsprecher ist eine Sekunde an und eine Sekunde aus. Suche dir selbst eine angenehme Frequenz.**

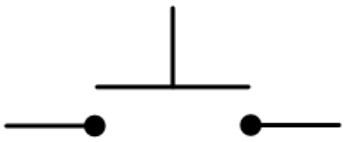
6. Der Taster

6.1 Verwendung des Tasters

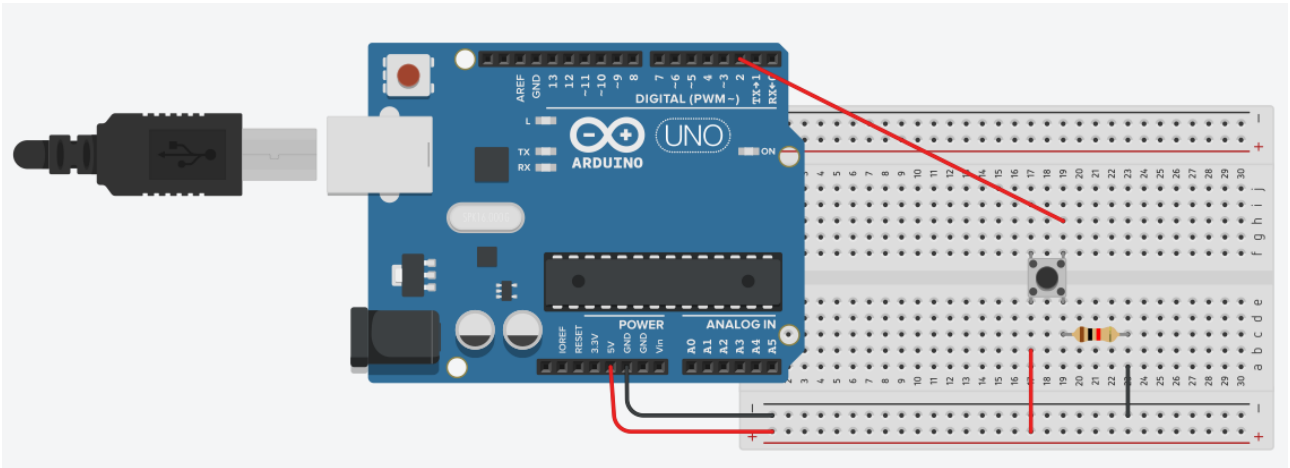
Bisher hast du die Pins des Mikrocontrollers als Ausgänge (`OUTPUT`) verwendet. Das Blinken der Leuchtdioden wurde nur vom übertragenen Programm gesteuert. Mit einem Taster kannst du nun "von außen" Einfluss nehmen. Einzelne Pins werden jetzt als Eingänge (`INPUT`) verwendet, die HIGH- und LOW-Signale bei gedrücktem oder nicht gedrücktem Taster an den Mikrocontroller senden.

➔ A6.1.1 **Licht wird mit einem Schalter ein- und ausgeschaltet. Eine Türklingel wird dagegen mit einem Taster betätigt. Worin besteht der Unterschied?**

Der Taster hat vier Anschlüsse um dem Druck deines Fingers besser standhalten zu können. Die beiden Pins auf der linken bzw. rechten Seite sind jedoch miteinander verbunden. Damit gibt es nur zwei Anschlüsse, die beim Drücken miteinander verbunden werden:



Man kann einen Taster dazu verwenden um dem Arduino ein Signal zu geben. Wird der Taster gedrückt, liegt am entsprechenden Pin des Arduinos ein hohes Potential an. Beim Loslassen ändert sich dies wieder zu niedrigem Potential. Die Änderung des Potentials kann der Arduino erkennen und je nach Programm einen Befehl ausführen.

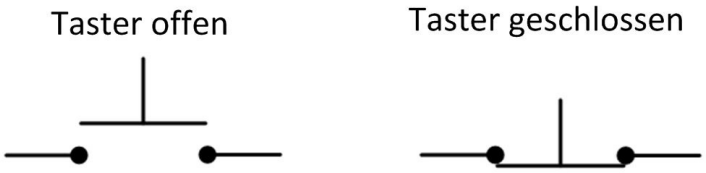


Über Pin 2 erhält der Arduino das Signal des Tasters. Ist der Taster nicht gedrückt, ist Pin 2 über den Widerstand (10kOhm) mit GND verbunden. Drückt man den Taster entsteht zusätzlich eine Verbindung von Pin 2 zu 5V. Pin 2 registriert jetzt das hohe Potential.

Der Widerstand zwischen GND und dem Taster verhindert einen Kurzschluss (=direkte Verbindung zwischen niedrigem und hohem Potential ohne Bauteil - es besteht die Gefahr der Zerstörung da sehr viel Strom fließt).
Eine solche Schaltung nennt man **Pull-Down-Widerstand**.

Beachte: Die Pins 0, 1 und 13 können beim Anschließen eines Tasters Probleme machen. Nutze stattdessen die anderen Pins.

➔ A6.1.2 a) **Zeichne den Schaltplan zum Pull-Down-Widerstand zweimal - einmal mit gedrücktem Schalter und einmal mit nichtgedrücktem Schalter. Markiere in den Schaltplänen jeweils Stellen gleichen Potentials mit gleicher Farbe.**



➔ A6.1.2 b) **Baue die Schaltung zum Pull-Down-Widerstand auf.**

6.2 Bedingte Anweisungen if - else

Mit Hilfe von Bedingungen können Programmabläufe programmiert werden, die auf Änderungen von außen reagieren. D.h. je nachdem welches Potential am Eingang anliegt, reagiert das Programm unterschiedlich.

Du kennst das selbst: In bestimmten Situationen im Alltag reagiert man je nachdem, ob eine Bedingung erfüllt ist, oder nicht, anders: "Wenn die Sonne scheint, gehe ich ins Schwimmbad." Du prüfst die Bedienung, indem du aus dem Fenster schaust und den aktuellen Zustand des Wetters abfragst. Im Programm kann eine solche Bedingungen z.B. sein, dass ein Wert größer (>), kleiner (<) oder gleich (==) einem anderen Wert ist. Abhängig davon, ob die Bedingung erfüllt ist (true) oder nicht (false), werden die Anweisungen ausgeführt oder nicht.

In unserem ersten Beispiel wählen wir als Bedingung, dass der am Pin 2 angeschlossene Taster gedrückt sein soll. Du hast gesehen, dass das Potential an Pin 2 je nach Taster-Stellung HIGH oder LOW sein kann. Der Zustand dieses Pins muss also abgefragt werden: Ist er HIGH oder LOW? Das passiert einmalig über den Befehl `digitalRead(Nummer des Pins);`. So wird festgestellt, ob der Schalter in genau diesem Moment gedrückt ist, oder nicht.

Dafür brauchst Du den folgenden Befehl:

```
if (Bedingung) {
      Anweisungen
}
```

Nach "`if`" steht immer eine Aussage, die sogenannte Bedingung. Wenn diese Bedingung erfüllt ist, das heißt, wenn die Aussage wahr ist, wird der Programmteil zwischen den geschweiften Klammern nach "`if`" ausgeführt.

➔ A6.2.1 **Schreibe in deinen Worten auf, was die folgende if-Abfrage aussagt.**

```
if (digitalRead(2) == HIGH) {
      digitalWrite(5, LOW);
}
```

Es kann auch sein, dass es mehrere Bedingungen gibt, die gleichzeitig erfüllt sein müssen (UND) oder von denen mindestens eine erfüllt sein muss (mathematisches ODER). Dafür schreibt man alle Bedingungen in die erste Klammer und verknüpft sie mit den Befehlen "`&&`" (UND) bzw. „`||`" (mathematisches ODER).

➔ A6.2.2 **Schreibe in deinen Worten auf, was die folgende if-Abfrage aussagt.**

```
if ((digitalRead(2) == HIGH) && (digitalRead(3) == HIGH)) {
      digitalWrite(5, HIGH);
}
```

Manchmal hast du einen Alternativ-Plan, zum Beispiel, wenn das Wetter schlecht ist: "Wenn die Sonne scheint, gehe ich ins Schwimmbad. *Sonst* gehe ich ins Kino." In diesem Fall benötigt ihr nach dem if-Befehl einen *else*-Befehl.

```
if (Bedingung) {
      Anweisungen
} else {
      Anweisungen
}
```

➔ A6.2.3 **Schreibe in deinen Worten auf, was die folgende if-Abfrage aussagt.**

```
if (digitalRead (2) == HIGH) {
      Serial.println("5V an Pin 2");
} else {
      Serial.println("0V an Pin 2");
}
```

Tipp: Wenn du dich nicht mehr an den Befehl `Serial.println` erinnerst, schau in Kapitel 1.3.2 nach!

Du weißt schon, wie man einen Taster so anschließt, dass man seinen Zustand ("gedrückt" oder "nicht gedrückt") auslesen kann. Bedingte Anweisungen kannst Du nun dafür nutzen, auf das Drücken eines Taster zu reagieren. Das folgende Programm lässt eine LED so lange leuchten, wie ein Taster gedrückt ist:

```
// ++++++++++ Festlegung der Konstanten ++++++++++++++++++++++++++++++++

const int buttonPin = 2;      // Pin 2 soll mit dem Taster verbunden sein

const int ledPin =  13;       /* Pin 13 ist mit der integrierten LED des
                         Arduino verbunden */

// ++++++++++ Festlegung der Variablen ++++++++++++++++++++++++++++++++


void setup() {


      pinMode(ledPin, OUTPUT);      //LED als Output festlegen

      pinMode(buttonPin, INPUT);    //Schalter als Input festlegen

}

void loop(){

      if (digitalRead(buttonPin) == HIGH) {     //Auslesen des Tasters:

            digitalWrite(ledPin, HIGH);
```

```
        } else {
              digitalWrite(ledPin, LOW);
        }
        /* Falls das Potenzial am Taster hoch ist, schalte die LED an, sonst
        schalte sie aus.*/
    }
```

➔ A6.2.4 **Probiere das Programm aus. Die Kommentare musst du nicht abtippen.**

➔ A6.2.5 **Ändere das Programm so ab, dass ein Lautsprecher im 1s-Takt piepst (d.h. pro Sekunde geht der Lautsprecher einmal an und einmal aus), falls der Schalter gedrückt ist.**

➔ A6.2.6 **Schreibe folgendes Programm: Wenn der Taster gedrückt wird, erscheint im Seriellen Monitor die Ausgabe "Taster ist gedrückt". Ist der Taster nicht gedrückt, erscheint im Seriellen Monitor "Warte auf Eingabe".**
Hinweis: *Informationen zur Anzeige im Seriellen Monitor findest du in Kapitel 1.3.2*

6.3 Anwendung: Ampel mit Taster

Fußgängerampeln schalten nur auf grün, wenn vorher ein Drücker betätig wurde.

➔ A6.3 a) **Öffne die Schaltung zur Ampel. Entferne die gelbe LED und schließe dafür an Pin2 einen Taster mit Pull-Down-Widerstand an.**

➔ A6.3 b) **Passe vor dem `setup`-Block die benötigten Konstanten an und lege Pin 2 im `setup`-Block als Eingang fest.**

➔ A6.3 **c) Schreibe ein Ampelprogramm, bei dem 20 Sekunden nach Drücken des Tasters die Ampel von rot auf grün schaltet. Nach 30 Sekunden soll die Ampel wieder auf rot springen, bis der Taster wieder gedrückt wird.**

6.4 Programme übersichtlich gestalten

Je länger deine Programme werden, umso schwieriger ist es für dich die Übersicht zu behalten. Wenn du die folgenden Punkte beachtest, kannst du dir die Arbeit viel einfacher machen und vor allem Fehler vermeiden:

- Trenne die einzelnen Programmblöcke durch deutlich sichtbare Strukturen wie:

```
// ########################################################
```

oder

```
// +++++++++++++++ GRUNDEINSTELLUNGEN ++++++++++++++++++++
```

- Verwende aussagekräftige Kommentare um zu beschreiben was im jeweiligen Programmteil passiert:

```
digitalRead(2);       //Lesen von Tasterstatus an Pin 2
```

- Rücke die Anweisungen nach jeder geschweiften Klammer etwas nach rechts ein rücke am Ende der geschweiften Klammer wieder nach links. Damit siehst du auf einen Blick ob du zu viele oder zu wenige geschweifte Klammern "öffnest" oder "schließt". Verwende diese Methode auch wenn du Schleifen in Schleifen programmierst!

alles linksbündig:

```
if (digitalRead(9) == HIGH) {
digitalWrite(2, HIGH);
if (digitalRead(10) == LOW {
digitalWrite(3, HIGH);}
} else {
digitalWrite(4, HIGH);}
```

mit Einrückungen:

```
if (digitalRead(9) == HIGH) {
   digitalWrite(2, HIGH);
   if (digitalRead(10) == LOW {
      digitalWrite(3, HIGH);
   }
} else {
   digitalWrite(4, HIGH);
}
```

➔ A6.4 **Beschreibe wie das Programm oben auf das Drücken von zwei Tastern reagiert. Vergleiche dabei die einfache Lesbarkeit der Versionen links und rechts.**

Ab jetzt: Gestalte deine Programme von Anfang an übersichtlich indem du die Regeln oben beachtest.

7. Zeitmessung

Für viele Anwendungen ist es notwendig, die Dauer zwischen zwei Ereignissen zu messen oder eine gewisse Zeitspanne vorzugeben, während der bestimmte Programmteile ablaufen sollen. Bisher kennst du schon den Befehl `delay(Zeit in Millisekunden);`, mit dem du eine zeitliche Abfolge steuern kannst. Während der Pause stoppt aber das gesamte Programm, d.h. es können auch keine Änderungen an den Input-Pins festgestellt und nicht darauf reagiert werden. Das Drücken eines an einen Input-Pin angeschlossenen Tasters während einer Pause ist beispielsweise wirkungslos. Dieses Problem kannst du mithilfe des Befehls `millis();` umgehen.

7.1 Zeitanzeige

Der Befehl `millis();` bestimmt die Zeit in Millisekunden, die seit dem Start des Programms vergangen ist. Wenn du diese angezeigt haben möchtest, benötigst du zusätzlich den Befehl `Serial.println();`, den du schon von der Textausgabe kennst. Anstelle eines Textes schreibst du nun `millis()` in die Klammer: `Serial.println(millis());`

➔ A7.1.1 a) **Schreibe ein Programm, das alle halbe Sekunde die Zeit, die seit Programmstart vergangen ist, in einer neuen Zeile am Serial Monitor anzeigt. Hinweis: Du darfst delay verwenden!**

➔ A7.1.1 b) **Was passiert, wenn du die Zeile** `Serial.println(millis());` **durch** `Serial.println("millis()");` **ersetzt?**

Auch Zeitabstände kannst du mithilfe des Befehls `millis();` messen. Dazu lässt du zu zwei verschiedenen Zeitpunkten die seit Programmstart vergangene Zeit bestimmen. Die Differenz der beiden Zeiten ist dann genau die Dauer vom ersten bis zum zweiten Zeitpunkt. Damit der Mikrocontroller diese Differenz berechnen kann, müssen die beiden gemessenen Zeiten zuerst mithilfe von Variablen zwischengespeichert werden.

Beachte bei der Wahl des Variablentyps, dass nicht beliebig große Zahlen gespeichert werden können. Die größte Zahl, die bei einer integer-Variable gespeichert werden kann, ist 2^15-1. Verwendet man anstelle von `int` den Typ `long`, so können Zahlen bis 2^31-1 gespeichert werden.

➔ A7.1.2 **Berechne, nach welcher Zeit der Speicher einer integer-Variablen, welcher der aktuelle millis( )-Wert zugeordnet wird, voll ist.**

Es ist also sinnvoll, für die Zeitmessung Variablen vom Typ `long` zu verwenden.

```
long zeit = millis();

/* Unter dem Namen "zeit" wird die bisher seit Programmstart
vergangene Zeit abgespeichert.*/
```

Für die Messung von Zeitabständen benötigst du Variablen, und zwar eine für die erste Zeit, eine für die zweite Zeit und eine für die Dauer.

Das Programm soll später folgendermaßen ablaufen: Die seit Programmstart vergangene Zeit wird gemessen und in der ersten Variablen gespeichert. Nach einer halben Sekunde wird wieder die aktuelle Zeit gemessen und unter der zweiten Variable gespeichert. Die Differenz zur Startzeit wird berechnet und als Dauer abgespeichert. Danach wird die Dauer am seriellen Monitor angezeigt. Dies soll sich in einer Dauerschleife wiederholen.

➔ A7.1.3 a) **Schreibe ein Programm für die Messung von Zeitdauern. Überlege dir dafür aussagekräftige Namen für die benötigten Variablen und definiere sie vor dem `setup`-Block.**

➔ A7.1.3 b) **Messe nach, wie lange es dauert, eine LED für 1 Sekunde an und für 1 Sekunde auszuschalten. Schreibe dafür das Zeitmessungs-Programm um.**

7.2 Blinken ohne delay

In diesem Abschnitt lernst du, wie du eine LED blinken lassen kannst, ohne den Befehl `delay();` zu verwenden. Dafür benötigst du Variablen, und zwar eine für das Potential an dem Pin, an dem die LED angeschlossen ist und eine für die zuletzt gemessene Zeit (hier: `referenzzeit`). Die aktuelle Zeit benötigt keine Benennung - sie ist immer in `millis()` gespeichert.

Außerdem kannst du die Intervalllänge für das Blinken, d.h. die Dauer, wie lange die LED jeweils an bzw. aus bleiben soll, als Konstante festlegen.

Die Grundidee eines Blinkprogramms ohne `delay();` ist die Folgende:

Startbedingungen: Die Variable `potenzial` hat den Wert LOW, der Variable `referenzzeit` wird `millis()` zugeordnet. Die Intervalllänge ist eine Konstante, die du ebenfalls vor dem Setup-Block definieren kannst (hier z.B. 500ms).

Mithilfe des `millis()`-Befehls wird nun immer wieder die aktuelle Zeit seit Programmstart abgefragt. Sobald die vorher festgelegte Intervalllänge überschritten ist (`millis() - referenzzeit > intervalllaenge`), passieren drei Dinge:

1. Die Variable `potenzial` wird von LOW auf HIGH gesetzt.
2. Das Potenzial am LED-Pin wird auf diesen neuen Potenzial-Wert gesetzt.
3. Der aktuelle `millis()`-Wert wird als neue Referenzzeit abgespeichert.

Nun wird wieder ständig der aktuelle `millis()`-Wert abgefragt und die seit der Referenzzeit vergangene Zeit berechnet. Sobald diese Zeit die Intervalllänge überschreitet, passieren wieder drei Dinge:

1. Die Variable `potenzial` wird von HIGH auf LOW gesetzt.
2. Das Potenzial am LED-Pin wird auf diesen neuen Potenzial-Wert gesetzt.
3. Der aktuelle `millis()`-Wert wird als neue Referenzzeit abgespeichert.

So geht das immer weiter. Jedes Mal, wenn die Intervalllänge überschritten wird, wird der Potenzial-Wert geändert und das Potenzial am LED-Pin auf den neuen Potenzialwert

umgeschaltet, d.h. abwechselnd von LOW auf HIGH und von HIGH auf LOW, und die aktuelle Zeit als Referenzzeit abgespeichert.

Du benötigst dafür also zwei ineinander geschachtelte If-Abfragen.

Mit der ersten (äußeren) if-Abfrage wird untersucht, ob die seit der letzten abgespeicherten Referenzzeit vergangene Zeit die Intervalllänge überschritten hat. Falls ja muss die aktuelle Zeit abgespeichert werden und das Potenzial gewechselt werden. Falls nein, passiert nichts.

Die zweite (innere) if-Abfrage benötigst du, um zu unterscheiden, ob von LOW auf HIGH oder von HIGH auf LOW umgeschaltet werden muss. Dazu wird der abgespeicherte Potenzial-Wert abgefragt. Falls er LOW ist, muss er auf HIGH gesetzt werden, sonst wird LOW als neuer Wert gespeichert.

➔ A7.2.1 **Schreibe ein Blink-Programm ohne delay().**

8. Schleifen

8.1 while-Schleife

Stelle dir folgende Situation vor: Du möchtest gerne Fahrrad fahren. Bei einem Blick nach draußen stellst du aber fest, dass es gerade regnet. Bei Regen bleibst du lieber in deinem Zimmer und liest ein Buch. Da du viel lieber Fahrrad fahren möchtest als lesen, schaust du nach jedem gelesenen Kapitel aus dem Fenster um zu überprüfen, ob es noch regnet. Wenn es regnet, liest du ein weiteres Kapitel, wenn es nicht regnet, gehst du Fahrrad fahren.

Diese Vorgehensweise entspricht dem Grundprinzip einer while-Schleife.
Eine while-Schleife hat die folgende Form:

```
while(Bedingungen) {
      Anweisungen;
}
```

Die Anweisungen in der geschweiften Klammer werden so lange wiederholt ausgeführt, wie die Bedingung erfüllt ist. Dies läuft folgendermaßen ab: Zuerst wird geprüft, ob alle Bedingungen erfüllt sind, falls ja, werden alle Anweisungen in der geschweiften Klammer ausgeführt. Anschließend springt das Programm wieder an den Anfang der Schleife. Es wird erneut geprüft, ob alle Bedingungen erfüllt sind, und falls ja, werden alle Anweisungen ausgeführt und das Programm springt wieder an den Anfang der Schleife. Dies passiert so lange, bis mindestens eine Bedingung nicht mehr erfüllt ist. In diesem Fall fährt das Programm mit den Anweisungen nach der Schleife fort.

Der "Programmcode" für die oben beschriebene Alltagssituation sähe also folgendermaßen aus:

```
while( Es regnet. ) {
      Lies ein Kapitel im Buch;
}
Geh Fahrrad fahren;
```

Bei folgenden Formulierungen bieten sich häufig while-Schleifen an: solange, sobald, während, bei Vorgabe einer bestimmten Dauer.

➔ A8.1.1 **Wie wird dieses Programm ablaufen? Kommt das Programm jemals in den loop-Block?**

```
int x=1;
void setup(){
 Serial.begin(9600);
 while(x<100){
        Serial.println(x);
        x=x+1;
   }
 while(2<5){
   Serial.print(„richtig“);
   }
}
void loop(){}
```

➔ A8.1.2 **Eine rote LED soll durchgehend leuchten bis ein Taster gedrückt wird. Danach sollen die rote LED und eine grüne LED abwechselnd im 1-Sekunden-Takt blinken. Schreibe das Programm.**

8.2 for-Schleife

Bei einer for-Schleife gibt es keine Bedingung. Stattdessen wird der Inhalt der Schleife mehrmals wiederholt und dann mit dem weiteren Programm fortgefahren. Die Anzahl der Wiederholungen legt man im Programmcode fest:

```
for(int i = 0; i< 10; i=i+1) {
      Lies ein Kapitel im Buch;
}
Geh Fahrrad fahren;
```

Bei der ersten Ausführung wird eine Variable zum Zählen der Durchläufe festgelegt und ihr ein Startwert zugeordnet. In unserem Beispiel ist die Variable eine ganze Zahl, hat den Namen `i` und den Anfangswert `0`. Bei jeder Durchführung der Schleife wird der Wert der Variable um eins erhöht. Die Schleife wird so lange ausgeführt, wie der Wert der Variable kleiner als 10 ist.

➔ A8.2.1 **Wie viele Kapitel werden aufgrund der oben stehenden Schleife im Buch gelesen?**

➔ A8.2.2 **Wie viele Kapitel würden im Buch gelesen werden, wenn am Anfang der Schleife der Wert der Variable auf 1 gesetzt werden würde?**

➔ A8.2.3 **Schließe sechs LEDs an aufeinanderfolgende Pins des Mikrocontrollers an und programmiere ein Lauflicht: Das heißt, die LEDs leuchten der Reihe nach für eine Sekunde auf. Benutze in deinem Programm eine for-Schleife!**

➔ A8.2.4 **Schließe eine rote LED und einen Taster an zwei Pins des Mikrocontrollers an. Immer, wenn der Taster gedrückt wird, soll daraufhin die LED 10 mal im 100 ms - Takt leuchten. Schreibe ein entsprechendes Programm.**

8.3 Fehler suchen und finden

Vielleicht hast du inzwischen festgestellt, dass es beim Programmieren nicht immer ganz einfach ist, einen Fehler zu finden. Schreibfehler wie fehlende Semikolons werden dir direkt angezeigt wenn du die Simulation startest (achte hierbei auf die Zeilenangabe für deinen Fehler!).
Schwieriger ist es, wenn dein Programm zwar formal richtig ist, aber nicht das macht was du möchtest. Mit folgender Vorgehensweise kannst du herausfinden, in welchem Programmteil sich der Mikrocontroller in jedem Moment befindet (z.B. nach einem Druck auf den Taster):

Verwende den Seriellen Monitor (siehe Kapitel 1.3.2) und notiere an den entscheidenden Stellen im Programm einen Befehl wie folgenden:

```
Serial.println("ich bin im Programmteil x");
```

Nun kannst du zum Beispiel herausfinden ob nach dem Drücken eines Tasters das Programm in die richtige Schleife wechselt.

Zusätzlich kannst du auch den aktuellen Wert deiner Variablen abfragen:

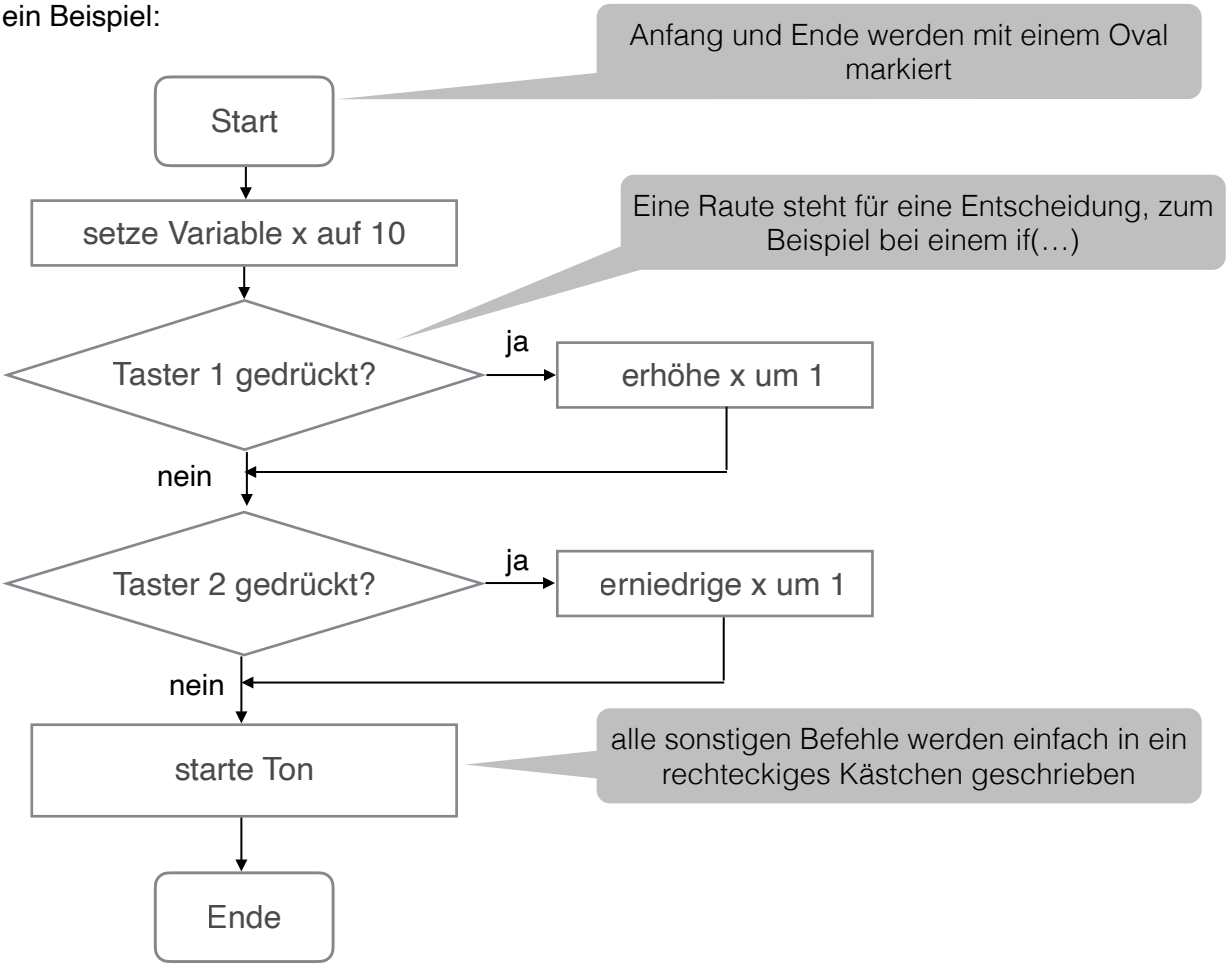
```
Serial.print("Variable potenzial");
Serial.println(potenzial);
```

➔ A8.3 **Notiere was du im Seriellen Monitor lesen kannst wenn das Programm die Abfrage der Variablen erreicht und** `potenzial` **den Wert HIGH gespeichert hat.**

9. Programme planen mit Programmablaufplänen

Bei längeren Programmen ist es sinnvoll, sich deren Ablauf zu überlegen, bevor man mit dem Programmieren beginnt. Dazu verwendet man Programmablaufpläne, die mit Hilfe weniger nützlicher Symbole das Programm strukturieren. Man kann sich ein Ablaufdiagramm wie ein Flussdiagramm vorstellen, in dem man sofort sieht, wo man eine Schleife oder eine if-Bedingung programmieren muss.

Hier ein Beispiel:



Beachte, dass sich die Pfeile, die den Ablauf zeigen, nie überkreuzen!

➔ A9.1 **Zeichne ein Programmablaufdiagramm zu folgendem Ablauf: Immer, wenn ein Taster betätigt wird, erscheint auf dem seriellen Monitor der aktuelle millis()-Wert.**

10. Unterprogramme

10.1 Unterprogramme ohne Variablen

Bei sich wiederholenden Programmteilen, zum Beispiel einem Refrain innerhalb einer Melodie, kann die Verwendung von Unterprogrammen Schreibarbeit sparen und gleichzeitig die Übersicht erhöhen.

Ein Unterprogramm ist ein Abschnitt in deinem Programmcode, der bei Bedarf immer wieder aufgerufen werden kann. Du kennst schon zwei Unterprogramme, sie heißen `setup()` und `loop()`. Das Unterprogramm `setup()` wird vom Mikrocontroller direkt nach dem Start einmal ausgeführt. Das Unterprogramm `loop()` wird direkt danach unendlich oft ausgeführt.

An deinen bisherigen Programmen kannst du erkennen:

- Vor dem Namen des Unterprogrammen muss das Codewort `void` stehen.
- Nach dem Namen des Unterprogramms müssen zwei runde Klammern „`()`“ stehen.
- Die zum Unterprogramm gehörenden Befehle werden von geschweiften „`{ }`“ Klammern eingefasst.

Der Befehl für das Unterprogramm „Refrain“ lautet:

```
void REFRAIN () {
     Anweisungen
}
```

Das Unterprogramm kann nun an jeder beliebigen Stelle in deinem Programm aufgerufen werden. Dies geschieht folgendermaßen:

```
REFRAIN ();
```

Nach Ablauf des Unterprogramms kann es folgendermaßen weitergehen:

1. Möglichkeit: Das Haupt-Programm läuft an der Stelle, an der das Unterprogramm aufgerufen wurde, weiter.
2. Möglichkeit: Am Ende des Unterprogramms wird ein anderes Unterprogramm aufgerufen.

In Programmablaufplänen wird für das Aufrufen eines Unterprogramms ein Rechteck mit zwei senkrechten Linien links und rechts verwendet:

Für jedes Unterprogramm wird ein eigener Programmablaufplan erstellt.

➔ A10.1.1 **Schließe 2 LEDs (grün und gelb) und einen Taster an die Pins des Arduinos an. Zu Beginn leuchtet die gelbe LED. Danach soll folgendes passieren:**

Immer wenn die gelbe LED leuchtet: Wenn der Taster gedrückt wird, geht die gelbe LED aus und dafür die grüne an.

Immer wenn die grüne LED leuchtet:
Wenn Taster gedrückt wird, geht die grüne LED aus und dafür die gelbe an.

Schreibe das Programm mit Hilfe zweier Unterprogramme.

10.2 Unterprogramme mit einer Variablen

An Unterprogramme kann man auch Werte für Variablen, so genannte Parameter, übergeben. Sie sagen dem Unterprogramm, was es genau machen soll. Der Befehl für das Unterprogramm lautet dann:

```
void UNTERPROGRAMMNAME (int VARIABLENNAME) {
      Anweisungen
}
```

So kann man beispielsweise mit einem Unterprogramm mehrere LEDs ansteuern:

```
void blinken(int pin){
  digitalWrite(pin, HIGH);
  delay(500);
  digitalWrite(pin, LOW);
  delay(500);
}
```

Die Variable pin steht hierbei für die jeweilige Pinnummer der LED, die zum Ansteuern der jeweiligen LED nur noch eingefügt werden muss.

➔ A10.2 **Schließe 3 LEDs an den Arduino an. Schreibe mit Hilfe des Unterprogramms „blinken“ ein Programm, dass diese nacheinander aufblinken lässt.**

10.3 Unterprogramme mit mehreren Variablen

Es ist auch möglich, das Unterprogramm von mehreren Variablen abhängig zu machen:

```
void blinken(int pin, int dauer){
      digitalWrite(pin, HIGH);
      delay(dauer);
      digitalWrite(pin, LOW);
      delay(dauer);
}
```

Möchte man die LED an Pin 4 für 10 Sekunden anschalten, benötigt man den Befehl `blinken(4, 10000);` um das entsprechende Unterprogramm aufzurufen.

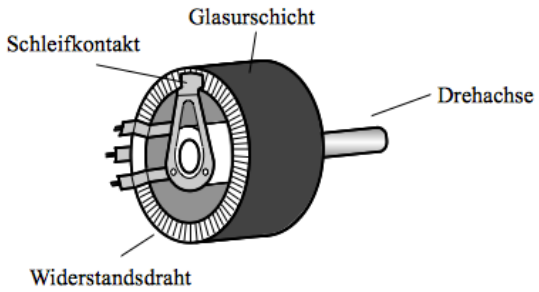
➔ A10.3 a) **Schließe 3 LEDs an den Arduino an. Schreibe mit Hilfe des Unterprogramms„blinken“ ein Programm, das diese nacheinander aufblinken lässt. Dabei soll die erste LED für 6 Sekunden leuchten, die zweite für 4 Sekunden und die dritte für 2 Sekunden.**

➔ A10.3 b) **Löse die vorherige Aufgabe mit Hilfe einer for-Schleife.**

11. Die analogen Pins

11.1 Das Potentiometer

Ein Potentiometer, kurz Poti genannt, ist ein Widerstand, dessen Wert verändert werden kann. Du hast dieses Bauteil sicherlich schon einmal benutzt - zum Beispiel wenn du die Lautstärke des Radios hoch und runter gedreht hast oder zum Dimmen des Lichts.
Dies funktioniert, indem über einen Widerstandskörper ein beweglicher Gleitkontakt (Schleifer) geführt wird. Das Potentiometer hat also drei Anschlüsse, zwei an den beiden Enden des Widerstandskörpers und dazwischen einen dritten für den Abgriff.



Der Schleifer teilt den Gesamtwiderstand in zwei in Reihe geschaltete Teilwiderstände auf. Legt man eine Spannung an den gesamten Widerstandskörper an, so teilt sich diese entsprechend der Teilwiderstände auf. Je nachdem, in welche Position der Kontakt gebracht wird, kann der Widerstand zwischen dem ersten Anschluss und dem Schleifer also alle Werte zwischen 0 Ohm und dem gesamten Widerstand annehmen. Die Teilspannung zwischen erstem Anschluss und Schleifer nimmt entsprechend alle Werte zwischen 0V und der Gesamtspannung an.

Die Welt des Arduinos war bisher digital - alles was er als Input erhalten oder als Output ausgegeben hat, war entweder HIGH (1) oder LOW (0). Um das Potentiometer mit seinen vielen möglichen Input-Werten nutzen zu können, braucht man daher die analogen Pins.

11.2 Analoge Pins

Der Befehl `analogRead(pinnummer)` liest das Potenzial eines festgelegten analogen Pins (A0-A5) mit einer 10 Bit Auflösung aus, d.h. es können 2^10=1024 verschiedene Werte gemessen werden. Jedem Potenzial von 0V bis 5V wird ein ganzzahliger Wert von 0 bis 1023 zugeordnet.
Dabei bedeutet
0 -> das Potenzial am analogen Pin beträgt 0V
1023 -> das maximale Potential von 5V liegt an

Die gemessenen Werte kann man beispielsweise mithilfe einer Variablen speichern:

```
int schleifer = analogRead(A0); /* liest den Wert am analogen Pin A0 aus
und weist ihn der Variablen "schleifer" zu */
```

Mit Hilfe der schon bekannten Befehle `Serial.print` und `Serial.println` kann der Messwert am Seriellen Monitor ausgegeben werden.

Analoge Pins müssen im Gegensatz zu digitalen **nicht** zuerst als Eingang oder Ausgang definiert werden.

Hinweis: Die analogen Pins können auch mit `digitalRead` als digitale Input-Pins verwendet werden.

➔ A11.2 a) **Schließe in der Simulation ein Potentiometer an den Arduino an. Der linke Anschluss wird mit GND und der rechte mit 5V verbunden. Am mittleren Anschluss, dem Schleifer, soll das Potenzial gemessen werden, er muss also mit einem analogen Pin verbunden werden.**

➔ A11.2 b) **Schreibe mit Hilfe des neuen Befehls ein Programm, welches dir am Seriellen Monitor den aktuellen Messwert am Schleifer des Potentiometers anzeigt.**

11.3 Potentiale berechnen

Wie bereits beschrieben, entspricht der ausgegebene Wert 0 einem Potential von 0V am Pin und der ausgegebene Wert 1023 dem maximalen Potential von 5V.

➔ A11.3.1 **Schreibe eine Formel auf, mit der man die ausgelesenen Werte so umrechnen kann, dass sie den Werten der am analogen Eingang anliegenden Potentiale entsprechen.**

Auch mit Variablen kann gerechnet werden. Die folgenden Beispiele zeigen dir, wie die Grundrechenarten Addition, Subtraktion, Multiplikation und Division durchgeführt werden.

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

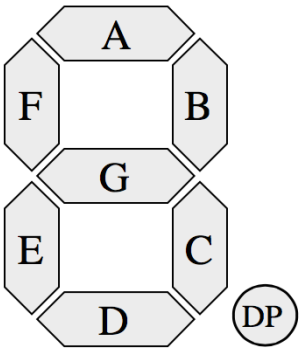
Bisher hast du den Variablentyp Integer verwendet, der nur ganzzahlige Werte zulässt. Nachkommastellen werden einfach abgeschnitten. Für die Berechnung der Potentiale ist eine Variable vom Typ `float` sinnvoll, unter der auch Dezimalbrüche gespeichert werden können.

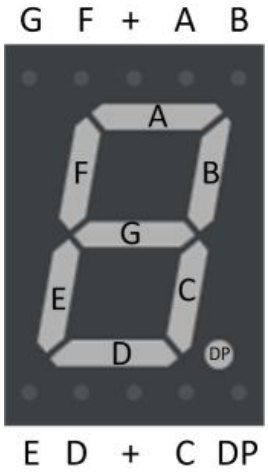
➔ A11.3.2 **Ändere dein Programm aus 11.2b) so ab, dass auf dem seriellen Monitor der Potentialwert ausgegeben wird, wie z.B.: „Das Potential beträgt: ... V“**

12. Die 7-Segment-Anzeige

12.1 Anschluss und Ansteuerung

7-Segment-Anzeigen finden Anwendung bei vielen technischen Geräten - zum Beispiel im Display eines Weckers, im Aufzug, um die Stockwerke anzuzeigen, oder auch im Taschenrechner. Sie bestehen aus 7 (teilweise auch aus 8) LED-Segmenten, die so angeordnet sind, dass man die Ziffern von 0 bis 9 mit ihnen darstellen kann. Diese Segmente sind folgendermaßen angeordnet und benannt:





➔ A12.1.1 **Färbe in deinen Unterlagen in einer leeren 7-Segment-Anzeigen jeweils die Segmente so, dass die entsprechende Ziffer dargestellt wird. Gib in einer Tabelle an, welche Segmente an (1) und welche ausgeschaltet (0) sein müssen.**

Hinter jedem Segment befindet sich eine LED. Beim Anschluss ist also auf die richtige Polung zu achten, außerdem muss ein Vorwiderstand verwendet werden. Es gibt 7-Segment-Anzeigen, bei denen alle LEDs eine gemeinsame Kathode haben und solche, die eine gemeinsame Anode haben. In der Simulation können beide Typen ausgewählt werden.

Wir verwenden in der Realität 7-Segment-Anzeigen mit gemeinsamer Anode, d.h. entweder der mittlere obere oder der mittlere untere Anschluss (siehe Abb. rechts) wird über den Vorwiderstand mit dem Powerpin 5V verbunden. Die Anschlüsse für die Segmente (A-G) und den Dezimalpunkt (DP) werden mit verschiedenen digitalen Pins verbunden, so dass du sie einzeln ansteuern kannst.

➔ A12.1.2 a) **Baue in der Simulation die Schaltung zur Ansteuerung der 7-Segment-Anzeige auf. Notiere dir, welcher Pin mit welchem Segment verbunden ist (Widerstand nicht vergessen!).**

➔ A12.1.2 b) **Schreibe ein Programm, das in einer Endlos-Schleife alle Segmente der Reihe nach aufblinken lässt.**
Tipp: Man steuert jedes Segment wie eine einzelne LED an. Auf welches Potential musst du die Pins also setzten, damit ein Segment leuchtet?

Nun soll ein Programm geschrieben werden, welches an der 7-Segment-Anzeige nacheinander die Zahlen 9 bis 0 ausgibt.

➔ A12.1.3 a) **Schreibe ein Programm, welches an der 7-Segment-Anzeige die 0 ausgibt.**

➔ A12.1.3 b) **Schreibe nun ein Programm, das von 9 auf 0 herunter zählt, also einen 10-Sekunden-Countdown.**

Insbesondere bei der 7-Segment-Anzeige kann man mit der Verwendung von Unterprogrammen Schreibarbeit sparen und gleichzeitig die Übersicht erhöhen.

Solch ein (unvollständiges) Unterprogramm könnte dann so aussehen:

```
void ZahlAnzeigen(int zahl) {
  if (zahl == 0) {
    digitalWrite(1, LOW);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(6, LOW);
    digitalWrite(7, HIGH);
  }
  if (zahl == 1) {
    digitalWrite(1, HIGH);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    digitalWrite(7, HIGH);
  }
}
```

➔ Zusatzaufgabe: Schreibe das Unterprogramm für alle Zahlen zu Ende und verwende es für deinen Countdown.

➔ Abschlussaufgabe: Vervollständige deine Übersicht für alle Programmier-Befehle und notiere dir die wichtigen Informationen zu allen Bauteilen (z.B. benötigter Widerstand, hohes / niedriges Potential), die du inzwischen kennengelernt hast.

Herzlichen Glückwunsch, du hast es geschafft :)!