Station 0 – Einstieg

Herzlich Willkommen zum Modul rund um das Thema Licht!

Ihr wollt heute also eine Einparkhilfe selbst konstruieren, Geschwindigkeiten messen, eine Blume mit der Sonne wandern lassen oder Temperaturen über Farben anzeigen? Das ist super! Bevor ihr euch allerdings auf diese Projekte stürzt, ist es wichtig, dass ihr erst einmal euer Material kennenlernt. Hier bekommt ihr einen Überblick über den Arduino-Mikrocontroller, die Bauelemente und die Programmierumgebung!

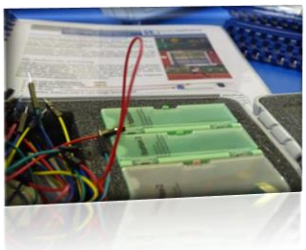
Während des gesamten Moduls geben euch die Arbeitsblätter Hinweise zur Umsetzung. Achtet dabei einfach auf die folgenden Symbole, die…

- euer Arbeiten strukturieren und Teilziele aufzeigen,
- euch Hilfen geben, Wichtiges, Schwieriges, etc. kennzeichnen und
- die Arbeitsaufträge und Aktionen beinhalten.

Ziele des Einstiegs

Wenn ihr diese Station geschafft habt, wisst ihr,…

- was mit dem Kram in der Box anzufangen ist.
- wie man eine LED zum Leuchten bringt.
- wie man eine LED blinken lässt.
- wie man einen Taster verwendet.



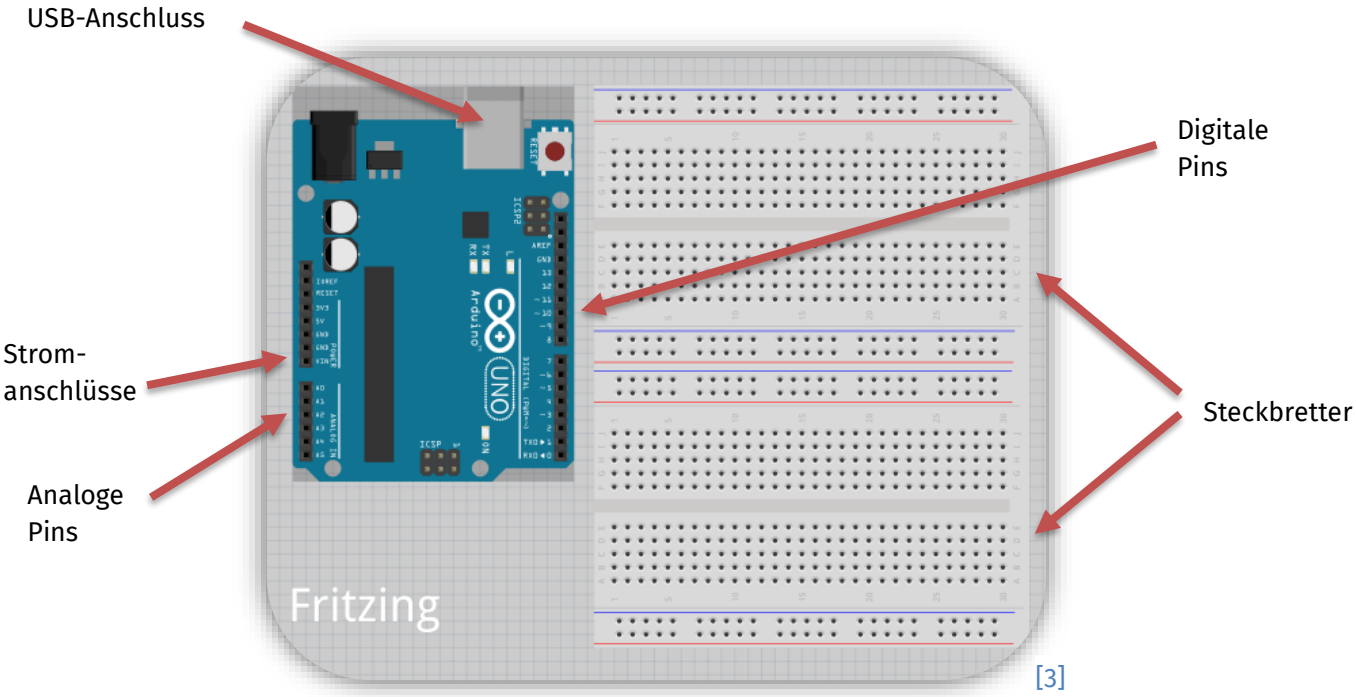
[2]

[1]

Dazu schaut ihr euch zunächst an, was wo angeschlossen wird, was bei LEDs zu beachten ist und wie man den Arduino programmiert.

Der Arduino

Als Erstes werdet ihr euch mit dem Arduino beschäftigen, der im Mittelpunkt jeder Schaltung steht.



[3]

Das ist nur eine schematische Darstellung, in der einige unbedeutende Teile weggelassen wurden. Selbst in dem Schema ist noch mehr zu sehen, als ihr wirklich braucht. Alles, was ihr wirklich benötigt, wird jetzt kurz erläutert:

Der USB-Anschluss

Hiermit verbindet ihr den Arduino später mit dem Computer, um euer Programm zu übertragen. Außerdem dient dieser Anschluss als Stromversorgung.

Die Stromanschlüsse

Manche Bauteile müssen mit Strom versorgt werden. Da Plus- und Minuspol nicht explizit als solche benannt sind, merkt euch Folgendes:

GND = – (Minuspol) 5V = + (Pluspol)

Bitte benutzt nur die Pins 2 bis 13. Die anderen Beiden haben eine Sonderfunktion.

Station 0 – Einstieg

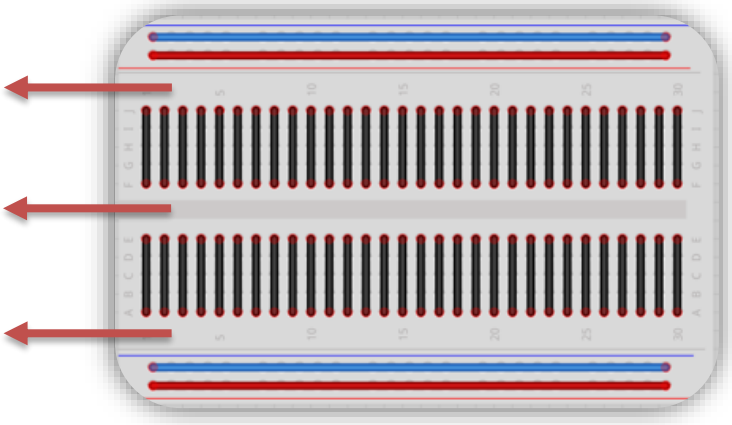
Die Pins

Im Gegensatz zu den **digitalen Pins**, werden die analogen Pins nur als Eingänge benutzt, daher spricht man auch von **analogen Eingängen**. Sie kennen nicht nur **HIGH** und **LOW**, sondern insgesamt 1024 verschiedene Werte. Sind 0V angeschlossen, ist der Wert 0, bei 5V ist er 1023. Doch dazu später mehr.

Die Steckbretter

Die weißen Platten mit den vielen Löchern bezeichnet man als Steckbretter. Die Löcher sind unter der Oberfläche miteinander verbunden. Wie genau seht ihr in der folgenden Abbildung:

Lücken, die ihr mit Kabeln überbrücken müsst

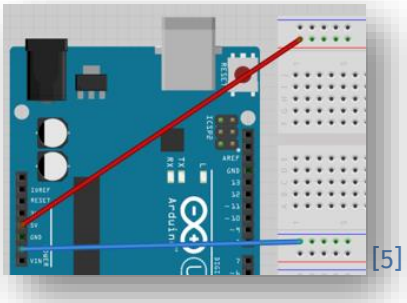


[4]

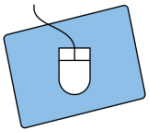
Bevor ihr etwas an der Verkabelung ändert, trennt den Arduino immer vom Strom! Zieht dazu einfach das USB-Kabel aus dem Computer.

Strom am Brett

Ein guter Start ist, erst einmal die äußeren Anschlussleisten am Steckbrett mit dem Arduino zu verbinden. So könnt ihr alle Bauelemente auf dem Steckbrett mit Strom versorgen, die äußeren Reihen sind also sowas wie eure Mehrfachsteckdosen.



[5]



Verbindet jetzt euren Arduino mit den zwei Steckkabeln (rot an 5V, blau an GND) mit dem Steckbrett.

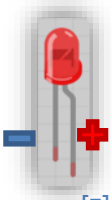
[6]

[***Tipp:*** *Das schwarze Kabel im Bild ist nur ein Beispiel. Ihr findet in eurem Kasten viele Kabel in verschiedenen Farben!*]

Station 0 – Einstieg

Es leuchtet

Bevor ihr programmieren werdet, lernt ihr nun, wie das mit der Verkabelung funktioniert und ob da wirklich schon Strom fließt. Außerdem lernt ihr zwei wichtige Bauteile kennen: **LED** und **Widerstand**.


[7]

Die LED

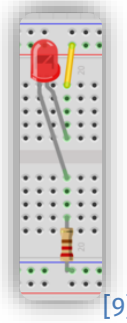
LED ist die englische Abkürzung für **Light Emitting Diode**. Es handelt sich also um ein Bauteil, das Licht aussendet. Dioden sind Bauteile, die Strom nur in eine Richtung durchlassen. Deswegen ist es sehr wichtig, dass sie richtig herum eingebaut werden. Beim Einbauen kann man sich grob an der Länge der Beinchen orientieren, denn eins ist **länger als das andere**. Das **Lange** muss mit dem **Pluspol** verbunden werden!

Leider sind LEDs übermäßig ehrgeizige Bauteile. Sie wollen immer so viel Licht wie möglich produzieren. Dabei erwärmen sie sich aber auch und werden im schlimmsten Fall so heiß, dass sie kaputtgehen.

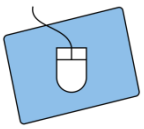
Deswegen muss der Strom immer begrenzt werden, wenn ihr mit LEDs arbeitet. Dafür gibt es **Widerstände**. In eurem Set findet ihr zwei Sorten. Die mit dem rot-rot-braun-goldenen Ringen (**220 Ω**) sind die Richtigen für diesen Fall.


[8]

Ob ihr den Widerstand vor oder nach der LED einbaut oder wie herum, spielt keine Rolle.

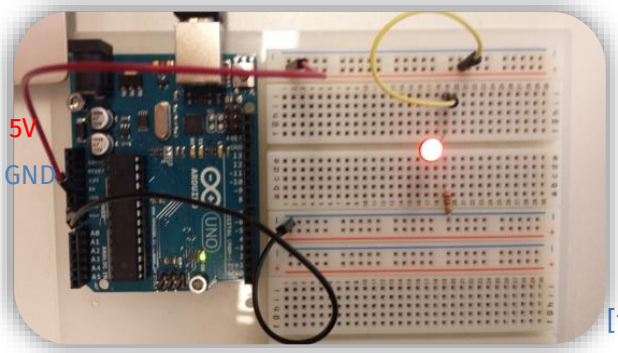

[9]

So sieht also das Schema zum Anschließen einer einzelnen LED aus.



Jetzt seid ihr dran: Nehmt ein gelbes Steckkabel, den Widerstand und die LED zur Hand und steckt das Ganze mal bei euch nach.
Zur Erinnerung: Das kurze Beinchen zeigt Richtung Minuspol!

Wenn ihr alles richtiggemacht habt, sollte euer Ergebnis so aussehen:



[10]

Falls eure LED **nicht** leuchtet:

- Ist das USB-Kabel eingesteckt?
- Steckt die LED richtig herum?
- Sind LED, Widerstand und Kabel in derselben Reihe?
- Habt ihr den richtigen Widerstand gewählt?

Station 0 – Einstieg

Das war zum Einstieg schon mal prima! Jetzt seid ihr bereit für die richtig spannenden Dinge wie die Programmierung des Mikrocontrollers. Auf dem Computer findet ihr ein Programm, das genauso heißt wie euer Mikrocontroller, nämlich Arduino.

Wenn ihr startet, seht ihr folgendes Fenster:

[11]

Bevor ihr loslegt, gibt es eine kurze Erklärung der wichtigsten Bedienelemente:

 [12] Mit diesen Knöpfen **überprüft** ihr euer Programm. Dabei macht euch die Software auf eventuelle Fehler im Code aufmerksam. Während der Linke nur überprüft, sendet der Rechte euer Programm nach erfolgreicher Überprüfung auch gleich an den Arduino.

Ihr müsst beim Senden des Codes an den Arduino ein wenig Geduld haben, da das Hochladen des Programms etwas dauern kann!

Beobachtet den **Ladebalken**, der rechts unten auftaucht und euch den Status anzeigt!

 [13] Dieser Knopf öffnet einen neuen Sketch.

 [14] Diese Knöpfe öffnen bzw. speichern Sketches.

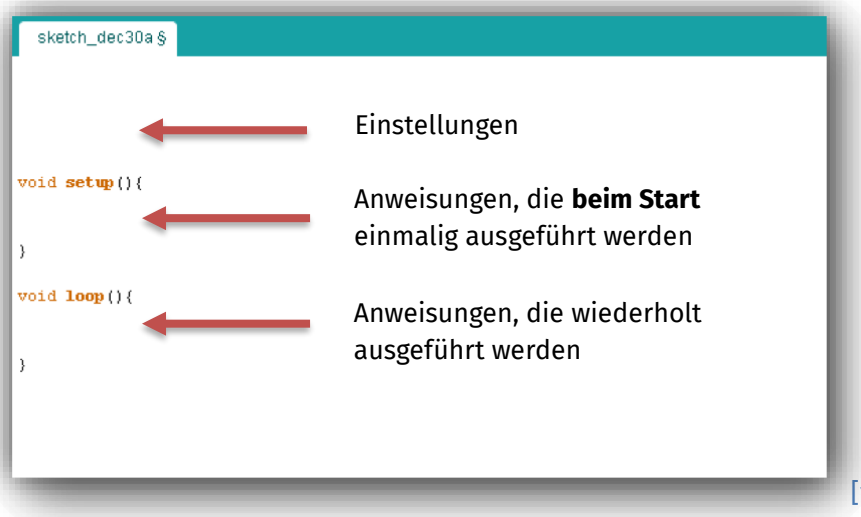
Sketch: Das ist einfach der Name für ein Programm in der Arduino-Sprache.

Station 0 – Einstieg

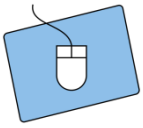
Erste Schritte in der Programmierung

Die Programmierung eures Arduinos ist ganz einfach. Es gibt ein paar Regeln, die ihr kennen und einhalten müsst, dann läuft der Rest „wie geschmiert".

So sieht das **Grundgerüst** aus, das ihr für jedes Programm brauchen werdet und das beim Anlegen eines neuen Sketches auch sofort erstellt wird:



[15]



Sollte dieses Grundgerüst doch nicht vorhanden sein, übertragt es in euren Sketch!

Anweisungen

Ihr gebt dem Arduino Anweisungen, was er zu tun hat. Damit der Arduino weiß, wann eine Anweisung zu Ende ist, schließt ihr jede Anweisung mit einem Semikolon (;) ab.

Variablen und Typen

Variablen sind kleine **Container**, in denen ihr Informationen ablegen könnt. Rechnet ihr etwas aus, könnt ihr das Ergebnis in einer Variablen (also einem Container) ablegen. Immer wenn ihr das Ergebnis braucht, müsst ihr nicht neu rechnen, sondern könnt einen Blick in den Container werfen.
Der Arduino muss aber vorher wissen, ob ihr z. B. Zahlen oder Wörter speichern wollt. So passen ganze Zahlen in Variablen vom Typen `int` (ausgeschrieben integer, engl. für ganze Zahl).
Wollt ihr eine neue Variable erzeugen (oder in Informatikersprache: **deklarieren**), schreibt ihr erst den Typ (also z. B. `int`), dann den Namen und schließt die Anweisung mit einem Semikolon ab:

```
typ name;
```

z. B. `int ergebnis;`

Bei euch im Programmcode werden Typen, die richtig geschrieben sind, auch orange gefärbt, wie hier im Beispiel.

Das Beispiel erzeugt eine Variable für ganze Zahlen, ihr Name ist „ergebnis“.
Ihr könnt auch direkt einen Wert speichern:

```
typ name = Wert;
```

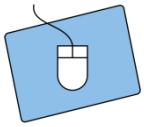
z. B.

```
int ergebnis = 13;
```

Der Arduino unterscheidet zwischen **Groß**- und **Kleinschreibung**! „Ergebnis“ ist also nicht gleich „ergebnis“!

Tipp: Entwicklerinnen und Entwickler können sich **Notizen** im Code machen. Alles hinter zwei Schrägstrichen (//) ist nur ein Kommentar und wird vom Programm ignoriert.

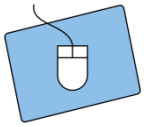
Das reicht erstmal an Theorie, jetzt geht es an die Programmierung!



Verändert zunächst eure Schaltung ein wenig, schließlich wollt ihr die LED jetzt mit eurem Arduino steuern.

1. Zieht dafür erstmal das **USB-Kabel** aus dem Computer.
2. **Verbindet** dann das obere Ende des gelben Steckkabels, das vorher in die Plus-Leiste ging, mit einem **digitalen Pin** des Arduinos. Notiert euch hier, wo ihr das Kabel eingesteckt habt: _____
3. Verbindet das **USB- Kabel** wieder mit dem Computer!

Jetzt, wo ihr den Arduino vorbereitet habt, könnt ihr mit der **Programmierung** anfangen!



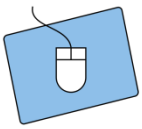
1. **Speichert** euren Sketch unter einem sinnvollen Namen im Arduino-Ordner (fragt hierzu eure Lehrkraft bzw. jemanden aus dem Betreuerteam, wenn ihr nicht wisst, wie). Das Grundgerüst habt ihr ja schon auf Seite 6 angelegt.
2. Dann legt ihr eine **Variable** an, die speichert, mit welchem Pin die LED verbunden ist. Noch einmal zur Erinnerung: Eine ganze Zahl braucht eine Variable vom Typ **`int`**. Also:

   ```
   int ledPin = ____ ;
   ```

   Diese Zeile kommt in den Bereich **Einstellungen** (siehe Seite 6).
   In die Lücke kommt die Nummer des Pins, in dem euer gelbes Kabel steckt.

Auf der nächsten Seite geht es weiter…

Station 0 – Einstieg



3. Nun zu **`setup()`:**
   a. `setup()` dient dazu, einige Grundeinstellungen vorzunehmen. Zum Beispiel, ob ein digitaler Pin ein Eingang oder Ausgang sein soll.
   b. Als Erstes muss also der Typ des Pins über folgende Anweisung angegeben werden:
   `pinMode(pin-name, pin-typ);`
   Tragt hier ein, wie die Zeile bei euch aussehen muss:
   `pinMode(_____, _____);`
4. Zum Herzstück: **`loop()`:**
   Eure LED ist mit einem digitalen Ausgang verbunden. Die Anweisung, um diesen entsprechend ein- oder auszuschalten, heißt
   `digitalWrite(pin-name, zustand);`
   Wie sieht die Zeile bei euch aus?
   `digitalWrite(_____, _____);`
5. Speichert euren Code. Führt ihn aus, indem ihr das USB-Kabel anschließt (falls ihr das noch nicht getan habt) und auf den Knopf zum Überprüfen und Übertragen klickt!

Statt `pin-name` tragt ihr den **Namen** ein, den ihr eurer Variablen gegeben habt (z. B. `ledPin`).
`pin-typ` ist hier `output`.

Zustand ist entweder `HIGH` für Strom an oder `LOW` für Strom aus.

Wenn ihr alles richtig gemacht habt, leuchtet eure LED jetzt wieder.

Hat es nicht geklappt? Schaut noch einmal über euren Code.

- Steht hinter jeder Anweisung ein Semikolon?
- Sind alle Klammern [() und {}] gesetzt?
- Enthält eure Variable die Nummer des korrekten Pins? Ist er überall gleich geschrieben?

Wenn es immer noch nicht klappt, vergleicht euren Sketch einmal mit folgender **Musterlösung**.

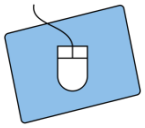
```
int ledPin = 8;  //Hier ist die LED an Pin 8 angeschlossen

void setup(){
  pinMode(ledPin,OUTPUT); //Pin 8 soll Ausgang sein
}

void loop(){
  digitalWrite(ledPin,HIGH);  //Einschalten
}
```

[16]

Wenn eure LED leuchtet, dann geht es auf der nächsten Seite mit etwas ganz Neuem weiter…

Station 0 – Einstieg

Jetzt lasst ihr die LED blinken! Die Anweisung dafür ist eigentlich ganz einfach: Da `loop()` immer und immer weiter läuft, dürfte es ja völlig ausreichen, die LED erst ein- und dann wieder auszuschalten.
[**Erinnerung**: Das Gegenteil von `HIGH` ist `LOW`.]



Passt euren Sketch entsprechend an und testet euer Programm.

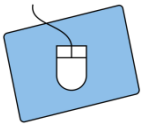
Huch, es leuchtet einfach weiter?! Nicht wirklich, es blinkt tatsächlich. Nur wird `loop()` so schnell ausgeführt, dass unser Auge gar nicht mitbekommt, dass die LED zwischendurch auch mal ausgeschaltet wird.

Zeit, den Arduino ein wenig auszubremsen. Dafür gibt es eine einfache Anweisung:

```
delay(zeit);
```

`delay(1000);` führt dazu, dass eine Sekunde Pause eingelegt wird.

Diese Anweisung lässt das Programm für die in Millisekunden angegebene `zeit` pausieren.



Fügt also einfach **`delays`** nach jedem Schaltvorgang ein und testet euren Sketch noch einmal.

So sollte es jetzt blinken:

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

[17]

Hervorragend, ihr könnt nun mit einem Mikrocontroller eine LED dazu bringen, zu blinken.

Licht auf Tastendruck!

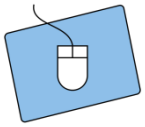
Nur Dinge ein- und auszuschalten, ohne von außen Einfluss darauf zu haben, ist irgendwie langweilig, oder? Deshalb lernt ihr jetzt ein neues Element kennen: den Taster.

Der Taster

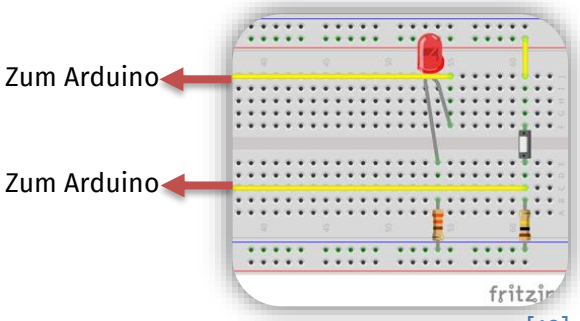
Ein Taster ist ein Bauelement, das Strom leitet, solange der Knopf runtergedrückt ist. Nimmt man den Finger weg, fließt auch kein Strom mehr.

Ihr findet einen Taster in eurer Box, er sieht so aus:  [18]

Ein paar Kleinigkeiten sind beim Einbau eines Tasters zu beachten. Die eine Seite des Tasters kommt an den Plus-Pol, die andere Seite wird mit einem der **digitalen Pins** des Arduinos verbunden. Zusätzlich muss diese Seite aber noch mit einem großen Widerstand (**100 kΩ,** braun-schwarz-gelb-gold) mit dem Minus-Pol verbunden werden.



Baut den Taster jetzt ein!
Eure Schaltung sollte dann so aussehen wie im Bild rechts:



[19]

Notiert hier, mit welchem **digitalen Pin** ihr den Taster verbunden habt: _____

Tipp: Versucht doch einmal, den Stromfluss vom Arduino durch die Schaltung und wieder zurück nachzuvollziehen!

Bevor ihr jetzt mit der Programmierung loslegt, müsst ihr zunächst einen **neuen Sketch** anlegen. Auch hier braucht ihr wieder das Grundgerüst von Seite 6, also `loop()` und `setup()`. Werft einen Blick auf euren letzten Sketch und kümmert euch schon mal um die **Variable für die LED**. Zwei Schritte sind hier wichtig: Die Variable zu **deklarieren** (im Bereich Einstellungen) und den Pin als **OUTPUT** (in `setup()`) zu setzen.

Auf der nächsten Seite lernt ihr ein weiteres wichtiges Konstrukt kennen, das ihr als angehende Programmierende bald im Schlaf berherrschen werdet.

Die if-Anweisung

```
1       if(bedingung){
2               ANWEISUNG1;
3               ANWEISUNG2;
…               ...;
        }
69      // nächste Zeile
```

Die **`if`-Anweisung** prüft, ob die angegebene `bedingung` **wahr** ist. Wenn ja, werden die Anweisungen in den **geschweiften Klammern** ausgeführt, wenn nicht werden sie ausgelassen und mit der *nächsten Zeile* (Sprung von Zeile 1 zu Zeile 69) im Programm weitergemacht.

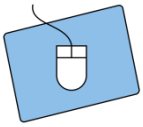
Doch wie wird so eine Bedingung formuliert? Zum Beispiel, indem ihr testet, ob der Taster gedrückt ist, also der Eingang den Wert **`HIGH`** meldet! Eine mögliche Bedingung wäre also:

```
if(digitalRead(tasterPin) == HIGH) {...Anweisungen;..}
```

So, wie ihr vorhin **`digitalWrite(pin-name)`** benutzt habt, um einen Ausgang zu benutzen, könnt ihr jetzt **`digitalRead(pin-name)`** verwenden, um einen digitalen Eingang zu benutzen.

Einige Dinge sollten euch hier auffallen:

1. Was ist denn `tasterPin`? So wie ihr euer Grundgerüst gerade mit den beiden Zeilen für den `ledPin` gefüllt habt, könnt ihr es jetzt zusätzlich auch mit der Variablen `tasterPin` ergänzen.
   Ihr habt dann also **zwei Variablen**: eine für die LED und eine für den Taster.
2. In **`setup()`** wird dem Arduino gesagt, ob der Pin ein Ein- oder Ausgang ist. Erinnert ihr euch noch, wie ihr das beim `ledPin` gemacht habt, der ein Ausgang war? Wie heißt dann entsprechend die Anweisung für einen Eingang?
   `pinMode(tasterPin, _____);`
3. Bisher kam immer nur ein einfaches „=“ vor. Plötzlich steht da aber ein doppeltes „==“. Das ist kein Tippfehler. Ein „=“ sagt dem Arduino, dass er einen **Wert zuweisen** soll. Mit „==“ **vergleicht** er beide Werte.



Versucht, die LED einzuschalten, sobald der Taster gedrückt wird.

Tipp: Um den Arduino neu zu starten, müsst ihr nicht jedes Mal das Kabel ziehen. Auf dem Arduino ist ein kleiner Knopf, der mit **Reset** beschriftet ist und das Gleiche tut.

[20]

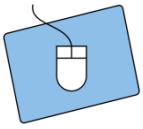
Und sonst?

Ein Schritt ist getan, aber ihr habt bestimmt auch etwas festgestellt: Die LED geht jetzt an, wenn ihr den Taster drückt, aber sie geht nicht wieder aus, wenn ihr loslasst.

Was fehlt, ist die Anweisung, was zu tun ist, wenn der Taster nicht gedrückt ist.
Dabei wird euch eine Erweiterung der `if`-Anweisung helfen:

```
if(Bedingung){
      ANWEISUNGEN;
}
else {
      ANDERE ANWEISUNGEN;
}
```

Den oberen Teil kennt ihr schon. Das englische Wort **`else`** steht für **sonst**. Die Anweisungen, die darauffolgen, werden nur dann ausgeführt, wenn die Bedingung **nicht erfüllt** wurde.



Passt euren Sketch so an, dass beim Loslassen des Tasters die LED wieder ausgeschaltet wird.

Arduino an Programmierende!

Der Arduino kann nicht nur über das Ein- und Ausschalten von LEDs mit euch kommunizieren. Gerade in den späteren Projekten wird es wichtig, dass ihr euch zwischendurch **Sensorwerte und Rechenergebnisse** auf dem Computer **anzeigen** lasst. Dafür haben die Entwickler von Arduino ein hervorragendes Werkzeug eingebaut: den Serial Monitor.

Der Serial Monitor

Der Serial Monitor zeigt euch Daten, die über das Kabel vom Arduino zum Computer geschickt wurden. Ihr startet diese Übertragung, indem ihr in **`setup()`** die Zeile

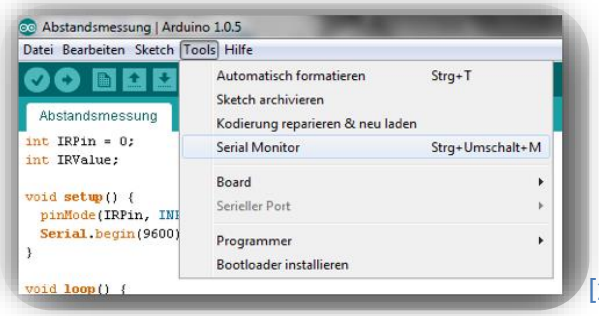
```
Serial.begin(9600);
```

einfügt. Die 9600 ist dabei die **Geschwindigkeit**, mit der eure Daten über USB gesendet werden. Wollt ihr Daten ausgeben, so könnt ihr sie in **`loop()`** über

```
Serial.println(daten)
```

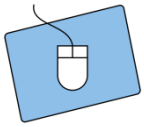
schicken. `daten` kann eine beliebige Variable sein, aber auch Text in Anführungszeichen, z. B. „LED eingeschaltet“.

Nach der Übertragung des Programms könnt ihr schließlich über den Reiter **Tools → Serial Monitor** (oder die entsprechende Tastenkombination) die Konsole in einem neuen Fenster öffnen.



[21]

Station 0 – Einstieg



Verändert euren Sketch jetzt so, dass nach jedem Schaltvorgang auch ein entsprechender Text ausgegeben wird.

Herzlichen Glückwunsch! Ihr habt alle Grundlagen gemeistert. Viel Spaß bei den Lichtprojekten!

Quellenverzeichnis:

Abb. 1, 2, 6, 10, 18, 20 – *Quelle: InfoSphere, CC BY-SA 4.0 Attribution-ShareAlike 4.0 International (https://creativecommons.org/licenses/by-sa/4.0/)*

Abb. 3 bis 5, 7 bis 9, 19 – *Quelle: Screenshots der Fritzing-Software (https://fritzing.org/), CC BY-SA 3.0 Attribution-ShareAlike 3.0 Unported (https://creativecommons.org/licenses/by-sa/3.0/), abgerufen am: 14.06.2022.*

Abb. 11 bis 17, 21 - *Quelle: Screenshots der Arduino-Software (https://www.arduino.cc/en/software), GNU Lesser General Public License (https://www.gnu.org/licenses/lgpl-3.0.de.html), abgerufen am 09.05.2022.*

 ,  ,  ,  *– Quelle: InfoSphere, CC BY-SA 4.0 Attribution-ShareAlike 4.0 International (https://creativecommons.org/licenses/by-sa/4.0/)*