Station 0 – Einstieg

Herzlich Willkommen zum Modul rund um das Thema Licht!

Ihr wollt heute also eine Einparkhilfe selbst konstruieren, einen Rollladen bedienen oder Temperaturen über Farben anzeigen?
Das ist super, aber bevor ihr euch auf diese Projekte stürzt, ist es wichtig, dass ihr erstmal euer Material kennenlernt. Hier bekommt ihr einen Überblick über den Arduino-Mikrocontroller, Tinkercad, 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...

- wie man Bauteile auswählt und anschließt.
- wie man eine LED zum Leuchten bringt.
- wie man eine LED blinken lässt.
- wie man einen Taster ein- und ausschaltet.

Abb. 1: InfoSphere-Kit

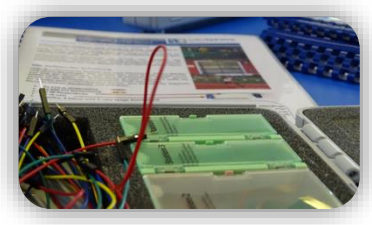


Abb. 2: Inhalt des InfoSphere-Kits

Damit wir anfangen können, müsst ihr zunächst folgende Schritte befolgen:

1. Klickt zunächst auf diesen Link: Tinkercad/arduino
2. Loggt euch nun ein. Wenn ihr dazu Hilfe braucht, fragt eine/n Betreuer/in.
3. Klickt nun auf Neuen Schaltkreis erstellen

CC BY SA

Der Arduino

Als Erstes schauen wir uns den Arduino an. Diesen findet ihr rechts bei den Komponenten unter Arduino Uno R3. Zieht ihn auf die Oberfläche links, und gebt ihm einen sinnvollen Namen (z. B. Arduino1 oder Hauptarduino).

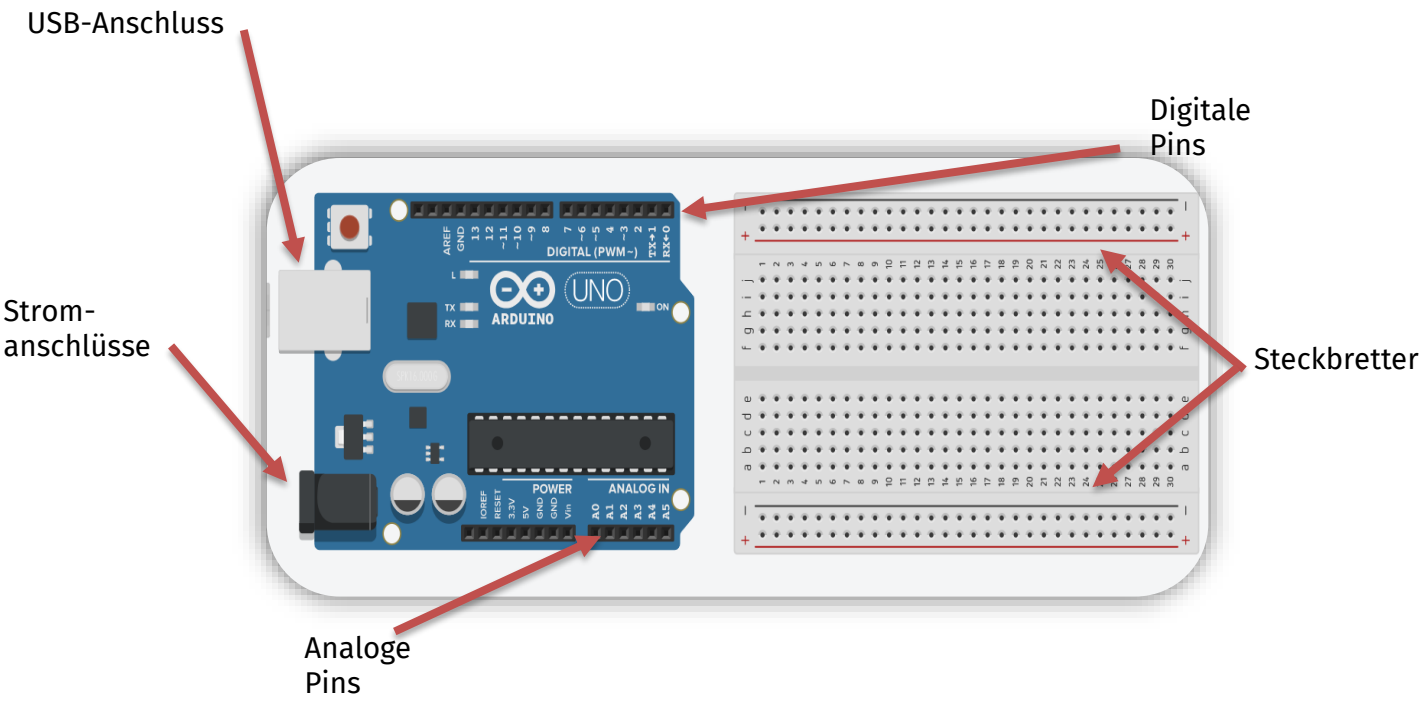


Abb. 3: Arduino-Board und Steckbretter

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

DER USB-ANSCHLUSS:

Damit verbindet ihr den Arduino später mit dem Computer, um euer Programm zu übertragen. Außerdem dient dieser Anschluss als Stromversorgung. Online ist unser Arduino natürlich immer mit Strom versorgt 😉.

STROMANSCHLÜSSE:

Manche Bauteile müssen mit Strom versorgt werden. Auch wenn es nicht explizit draufsteht, merkt euch einfach:

GND = **– (Minuspol)** 5V = **+ (Pluspol)**

⚠ Bitte benutzt nur die digitalen Pins 2 bis 13. Die anderen beiden haben eine Sonderfunktion!

CC BY SA

Station 0 – Einstieg

Im Gegensatz zu den digitalen Pins, werden die analogen Pins nur als Eingänge benutzt, daher sprechen wir auch von den **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.

Wenn ihr alles auf eurem Arduino gefunden habt, geht's an die Steckplatine.

Auf den **STECKPLATINEN** seht ihr viele Löcher. Diese sind unter der Oberfläche miteinander verbunden. Das markiert Tinkercad auch, wenn ihr den Mauszeiger über die Löcher haltet. So sieht das Ganze unter der Oberfläche aus:

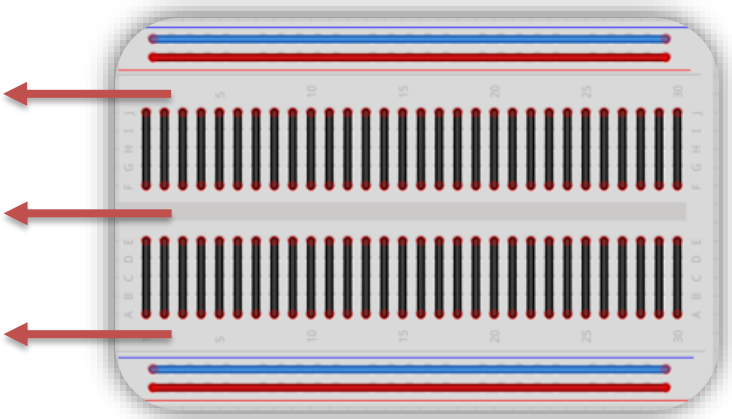


Abb. 4: Steckbrett mit gekennzeichneten Verbindungen

STROM AM BRETT

Ein guter Start ist es, 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.

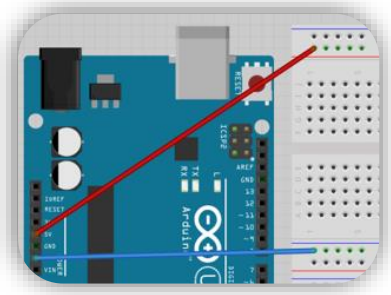
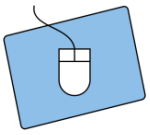


Abb. 5: Stromanschlüsse



Verbindet jetzt euren Arduino mit den zwei Steckkabeln (rot an 5V, blau an GND) mit dem Steckbrett. Befolgt dazu die folgenden Schritte:

1. Zieht ein Steckbrett von den Komponenten auf die Oberfläche.
2. Klickt dazu zunächst auf die eine Stelle, die durch das Kabel verbunden werden soll, und dann auf die andere.

Abb. 6: Steckkabel

CC BY SA

Station 0 – Einstieg

Es leuchtet

Bevor es ans Programmieren geht, werdet ihr schonmal lernen, wie das mit der Verkabelung funktioniert und ob da wirklich schon Strom fließt. Außerdem lernt ihr zwei wichtige Bauteile kennen: **LED** und **WIDERSTAND.**

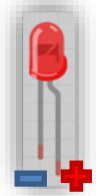


Abb. 7: LED

LEDs

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. Zum Einbauen kann man sich an den Beinchen orientieren, denn *eins ist gekrümmt (und länger) und das andere gerade*. Das gekrümmte Bein muss mit dem **PLUSPOL** verbunden werden!

Leider sind LEDs übermäßig ehrgeizige Bauteile. Sie produzieren immer so viel Licht wie möglich! In einer realen Schaltung würden sie schnell kaputtgehen.

Deswegen muss der Strom immer begrenzt werden, wenn ihr mit LEDs arbeitet. Dafür gibt es **WIDERSTÄNDE**. Widerstände findet ihr rechts bei den Komponenten. Zieht einen Widerstand rüber, und verkabelt die LED wieder mit dem Strom, allerdings dieses Mal mit einem Widerstand von 220 Ω dazwischen (klickt zum Umstellen einfach auf den Widerstand). Wenn ihr alles richtiggemacht habt, erscheint keine Fehlermeldung an der LED.

Abb. 8: Widerstand

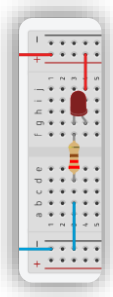
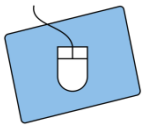


Abb. 9: LED mit Widerstand

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



Jetzt seid ihr dran. Nehmt den Widerstand und die LED zur Hand und steckt das Ganze mal bei euch nach.
Zur Erinnerung: Das lange, gekrümmte Beinchen zeigt Richtung Pluspol!

Wenn ihr alles richtig gemacht habt, sollte euer Ergebnis so aussehen:

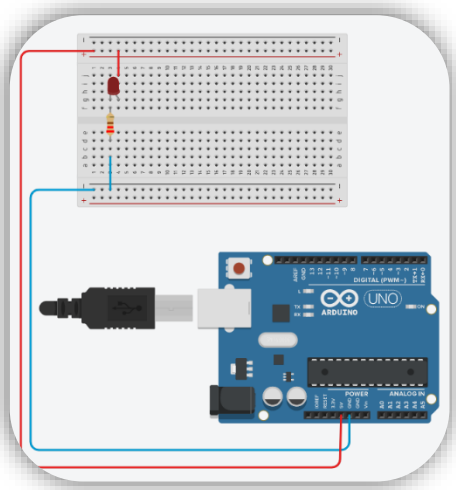


Abb. 10: Komplette Schaltung

Klickt nun auf „Simulation starten“. Falls eure LED nicht leuchtet:

- Leuchtet ein i oder „?“ bei der LED auf?
- Steckt die LED richtig herum?
- Sind LED, Widerstand und Kabel in der gleichen Reihe?
- Habt ihr den richtigen Widerstand eingestellt und auf Ω gestellt?

CC BY SA

Das war zum Einstieg schon mal prima! Jetzt seid ihr bereit für die richtig spannenden Dinge wie die Programmierung des Mikrocontrollers. Unter Code könnt ihr einen Editor öffnen. Klickt auf den Button, und wählt anschließend links oben Text aus (dort steht entweder Text und Blöcke, Blöcke oder Text).

Nun sollte euer Fenster rechts so aussehen:

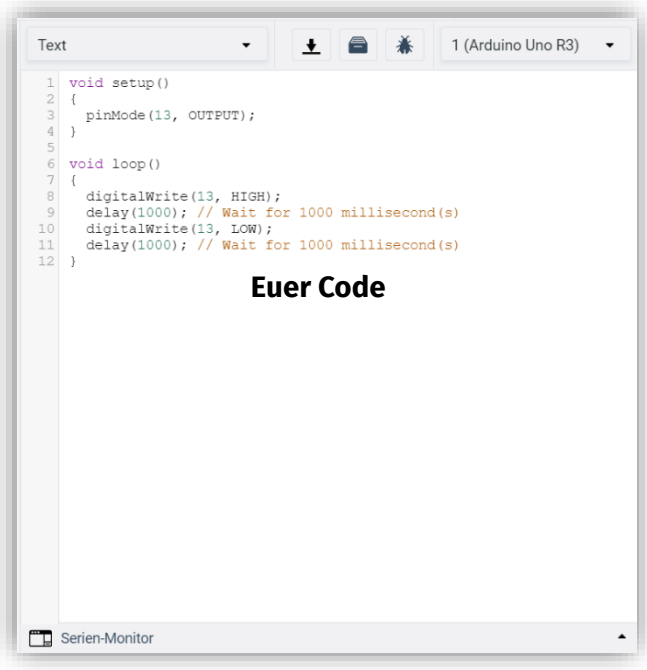


Abb. 11: Screenshot Tnkercad Codeansicht

Bevor ihr loslegt, gibt es noch eine kurze Erklärung zu einem wichtigen Button:

Mit diesem Knöpfchen **ÜBERPRÜFT** ihr euer Programm. Dabei macht euch die Software auf eventuelle Fehler im Code aufmerksam.

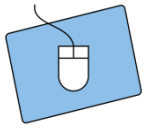
CC BY SA

Programmierung: Erste Schritte

Das Programmieren 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. Den übrigen Code könnt ihr löschen.

```
1  // hier kommen eure Einstellungen hin
2  void setup()
3  {
4    // hier kommen Anweisungen hin, die beim Start ausgeführt werden
5  }
6
7  void loop()
8  {
9      // hier kommen Anweisungen hin, die wiederholte ausgeführt
10     // werden sollen.
11 }
```

Abb. 12: Aufbau eines Arduino-Sketches



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

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 Variable (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 (hier im Modul immer **INT**) auf, 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 lila gefärbt wie hier im Beispiel.

Das Beispiel erzeugt eine Variable für ganze Zahlen, ihr Name ist „ergebnis“.
Ihr könnt aber 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“!

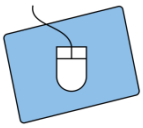
[***Tipp:*** *Entwickler können sich auch Notizen im Code machen. Alles hinter zwei Schrägstrichen (//) ist nur ein Kommentar und wird vom Arduino ignoriert.*]

CC BY SA

Station 0 – Einstieg

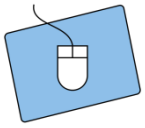
Genug Theorie, jetzt geht es an die Programmierung!

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



1. Stoppt dazu zunächst die Simulation.
2. **VERBINDET** dann einfach das obere Ende des Steckkabels, das vorher in die Plus-Leiste ging, mit einem **DIGITALEN PIN** des Arduino.
   Notiert euch hier, wo ihr das Kabel eingesteckt habt: _____

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



1. Legt eine **VARIABLE** an, die speichert, mit welchem Pin die LED verbunden ist. Noch einmal zur Erinnerung: Eine ganze Zahl kommt in 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 Kabel steckt.
2. Nun zu setup():
   a. setup() dient dazu, einige Grundeinstellungen vorzunehmen. Zum Beispiel, ob ein digitaler Pin ein Eingang oder Ausgang sein soll.
   b. Legt also als Erstes den Typ des Pins fest. Nutzt dazu die Anweisung
   pinMode(pin-name, **PIN-TYP**);
   Tragt hier ein, wie die Zeile bei euch aussehen muss:
   pinMode(_____,_____**)**;
3. 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(_____, _____**)**;
4. Führt euren Code nun aus, indem ihr auf „Simulation starten“ drückt.

Anstatt pin-name tragt ihr den Namen ein, den ihr eurer Variable 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, ob hinter jeder Anweisung ein Semikolon steht, alle Klammern [() und {}] korrekt gesetzt sind, eure Variable die Nummer des richtigen Pins enthält und auch überall gleich geschrieben ist.

CC BY SA

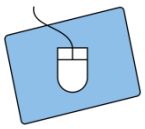
Wenn es immer noch nicht klappt, vergleicht euren Sketch einmal mit folgender Musterlösung.

```
int ledpin = 6; //Hier ist die LED an Pin 6 angeschlossen

void setup()
{
  pinMode(ledpin,OUTPUT); //Pin 6 soll ein Ausgang sein
}

void loop()
{
  digitalWrite(6, HIGH); //Pin 6 soll eingeschaltet werden
}
```

Abb. 13: Programmcode zum Einschalten einer LED

Prima, die LED leuchtet wieder. Jetzt geht es direkt weiter mit etwas ganz Neuem: *Lasst sie blinken!*

Die Anweisung dafür ist eigentlich ganz einfach: Da loop() ja immer und immer weiterläuft, 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.

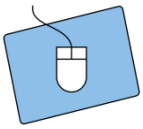
Huch, es leuchtet einfach weiter? Nicht wirklich, es blinkt tatsächlich. Nur wird loop() so schnell ausgeführt, dass unser Auge gar nicht mitbekommt, dass die LED zwischendurch auch mal ausgeschaltet wird.

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

delay(zeit**)**;

Diese Anweisung lässt das Programm für genau zeit Millisekunden pausieren.

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



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

HERVORRAGEND, ihr könnt jetzt also mit einem Mikrocontroller eine LED zum Blinken bringen!

CC BY SA

So sollte es jetzt blinken:

```
void loop()
{
    digitalWrite(ledPin, HIGH); //Pin einschalten
    delay(1000);
    digitalWrite(ledPin, LOW); //Pin ausschalten
    delay(1000);
}
```

Abb. 14: Programmcode zum Blinken einer LED

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.**

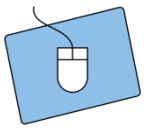
DER TASTER

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

Ihr findet einen Drucktaster unter Komponenten; er sieht so aus:

Abb. 15: Taster

Ein paar Kleinigkeiten sind beim Einbau eines Tasters jedoch zu beachten. Die eine Seite des Tasters kommt an den Plus-Pol, die andere Seite wird mit einem der digitalen Pins des Arduino verbunden. Zusätzlich muss diese Seite aber noch mit einem großen Widerstand (**100 kΩ**) mit dem Minus-Pol verbunden werden. Nutzt dazu entweder 1a und 2a oder 1b und 2b.
*[**HINWEIS:** Eine eurer Plusleisten ist an den ledPin angeschlossen, für den Taster benötigt ihr eine weitere Plusleiste.]*



Baut den Taster jetzt ein!

Eure Schaltung sollte dann so aussehen wie rechts auf dem Bild:

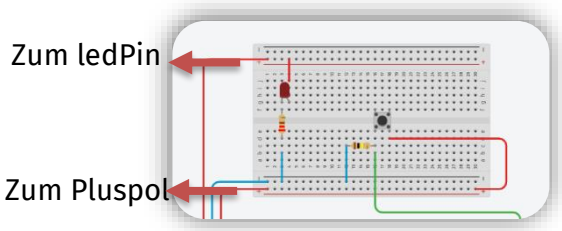


Abb. 16: Schaltung mit Taster und LED

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!*]

CC BY SA

Station 0 – Einstieg

Bevor ihr jetzt mit dem Programmieren loslegt, müsst ihr zunächst einen **SKETCH** leeren. 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 wichtig: Die Variable **DEKLARIEREN** (im Bereich Einstellungen) und den Pin als **OUTPUT** (in setup()) setzen.

Ihr lernt jetzt ein weiteres wichtiges Konstrukt kennen, das ihr als angehende Programmierende bald im Schlaf berherrschen werdet.

DIE IF-ANSWEISUNG

```
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, dann werden sie ausgelassen und mit der **NÄCHSTEN ZEILE (SPRUNG VON ZEILE 1 ZU ZEILE 69)** im Programm weitergemacht.

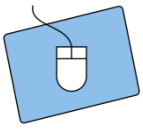
Doch wie wird so eine Bedingung formuliert? Beispielsweise, indem ihr testet, ob der Taster gedrückt ist, also der Eingang den Wert **HIGH** meldet! Eine mögliche Bedingung wäre also:

```
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? Ja, gut aufgepasst! So wie ihr euer Grundgerüst gerade mit den beiden Zeilen für den ledPin gefüllt habt, könnt ihr es jetzt zusätzlich auch mit der Variable tasterPin ergänzen. Ihr habt dann also **ZWEI VARIABLEN**: eine für die LED und eine für den Taster.
2. In **setup()** wird dem Arduino ja 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 vielleicht die Anweisung für einen Eingang? **pinMode(**tasterPin, _____**);**
3. Vorher kam immer nur ein einfaches „=“ vor. Plötzlich steht da aber ein doppeltes „==“. Das ist kein Tippfehler. Ein „=“ sagt dem Arduino **WEISE EINEN WERT ZU**, während „==“ ihm mitteilt **VERGLEICHE BEIDE WERTE**.



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

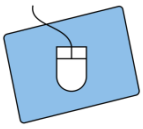
Und sonst?
Ein Schritt ist getan, aber ihr habt bestimmt auch etwas festgestellt: Die LED geht jetzt an, wenn ihr den Taster drückt, aber sie geht nicht wieder aus, wenn ihr loslasst!

CC BY SA

Was fehlt, das ist die Anweisung an den Arduino, 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 ja schon. Das englische Wort else steht für **SONST**. Die Anweisungen, die darauffolgen, werden nur dann ausgeführt, wenn die Bedingung nicht erfüllt wurde.



Schafft ihr es, euren Sketch so anzupassen, dass beim Loslassen des Tasters die LED wieder ausgeschaltet wird?

Arduino an Programmierenden!

Der Arduino kann nicht nur über das Ein- und Ausschalten von LEDs mit dir 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.

DER SERIEN MONITOR

Der Serien 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“.

Wenn ihr die Daten jetzt sehen wollt, könnt ihr unter Code unten auf „Serien Monitor“ klicken und euch so die Daten anzeigen lassen.

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, 15, 17 – *Quelle: InfoSphere*
Abb. 3, 4, 5, 7, 8, 9, 16 – *Quelle: Screenshots von Tinkercad* https://www.tinkercad.com/
, , , angefertigt vom InfoSphere-Team

CC BY SA