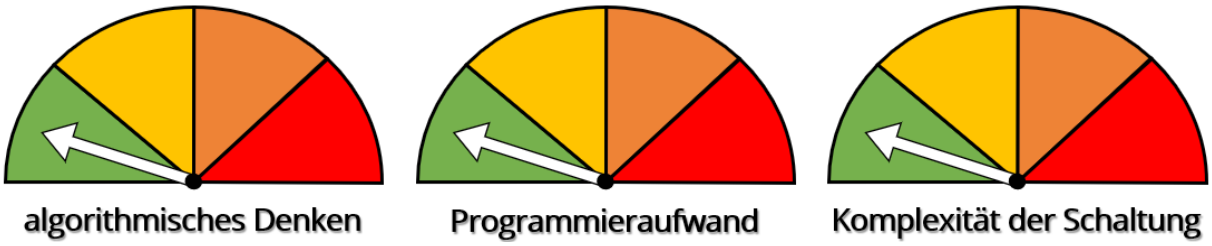
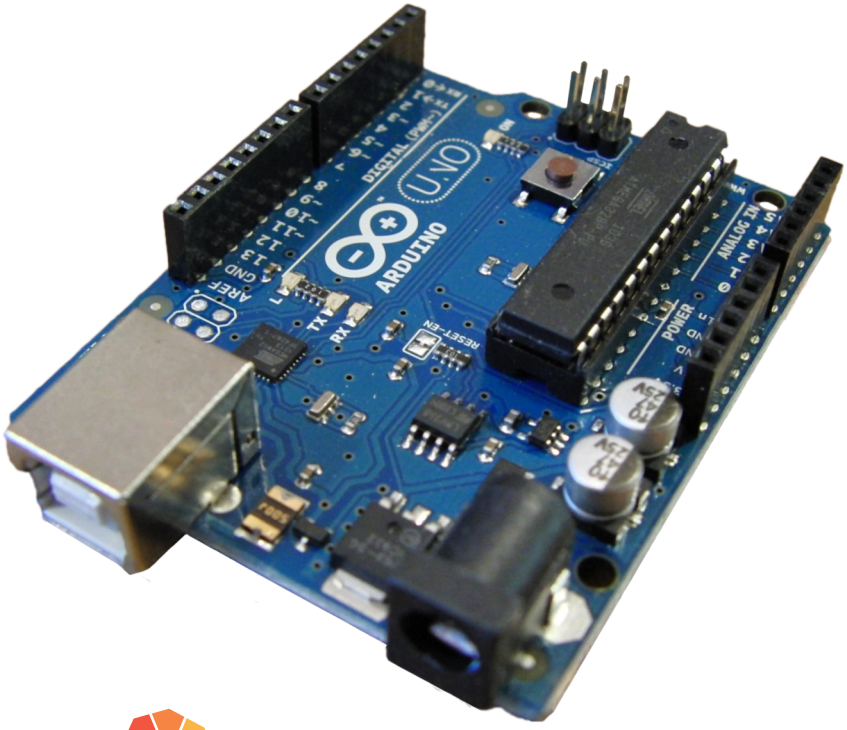
TECHNISCHE UNIVERSITÄT DRESDEN

Arduino Uno

Einstieg | „Hallo Welt“ des Physical Computing





Professur für Didaktik der Informatik

SYLBER Synergetische Lehrerbildung im exzellenten Rahmen

Das Maßnahmenpaket „TUD-Sylber² – Synergetische Lehrerbildung im exzellenten Rahmen“ wird im Rahmen der gemeinsamen „Qualitätsoffensive Lehrerbildung“ von Bund und Ländern aus Mitteln des Bundesministeriums für Bildung und Forschung gefördert.

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

EINSTIEG IN DIE PROGRAMMIERUNG MIT DEM ARDUINO UNO

Aller Anfang ist schwer – wir wollen es dir heute aber hoffentlich besonders einfach machen, um deine ersten Programmiererfahrungen mit einem Microcontroller zu machen. Im Rahmen des Workshops stehen dir die verschiedensten Station zur Verfügung. Du kannst dabei zum Beispiel eine Temperaturmessung programmieren, Geschwindigkeiten berechnen, oder ein Reaktionsspiel entwickeln. Dafür benötigst du wichtige Grundlagen, welche du in dieser Station lernen wirst.

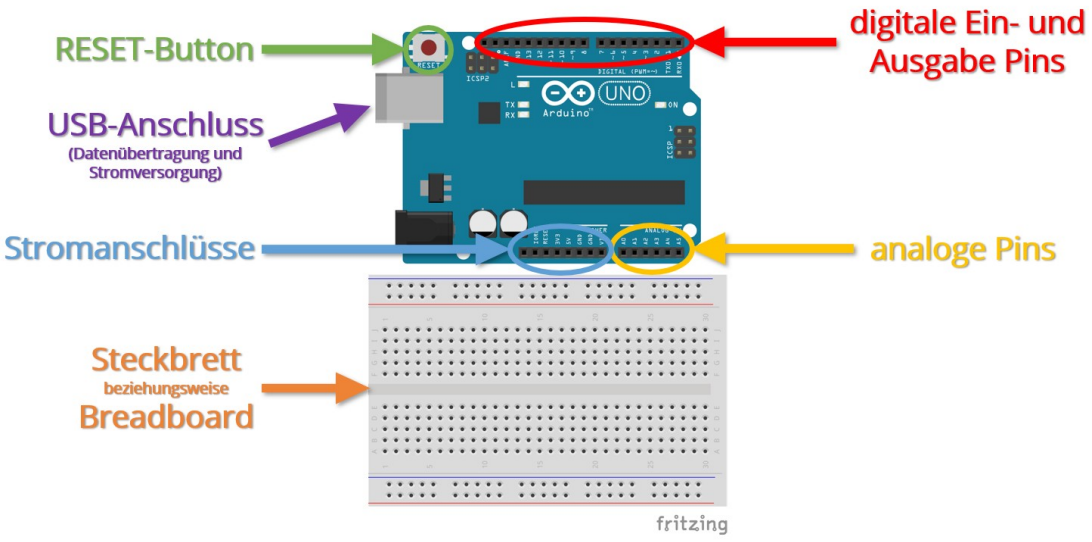
UM WAS WIRD ES IN DIESER STATION GEHEN?

Zuerst werden wir uns den Arduino genauer anschauen. Wir werden lernen, wie wir beispielsweise LEDs anschließen und was wir dabei beachten müssen. Außerdem lernst du das sogenannte Breadboard kennen und wirst dir das Tool zum Programmieren genauer anschauen!

Hallo Welt! - Schon lange ist dies das erste, das Programmiererinnen und Programmierer in einer neuen Programmiersprache ausprobieren. Beim Physical Computing – die Programmierung physischer Systeme – ist dieses „Hallo Welt“ das Ansteuern einer LED.

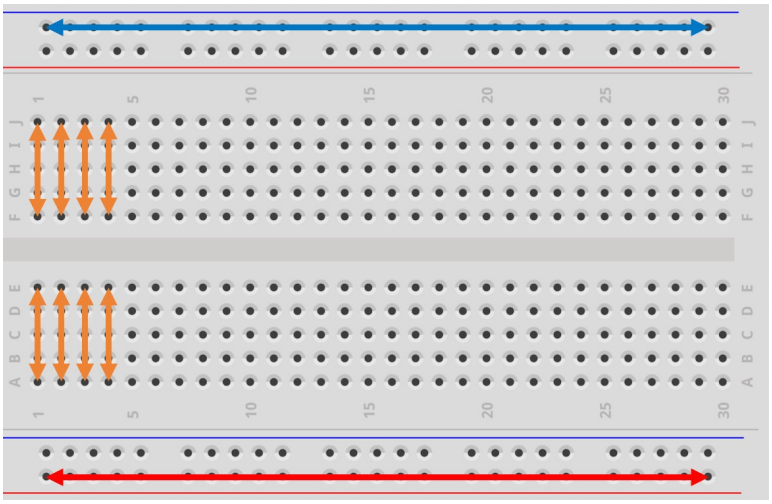
An dieser Station sollst du deshalb eine LED zum Leuchten bringen! Wenn du das geschafft hast, erweiterst du das Projekt so, dass die LED dauerhaft blinkt. Da eine blinkende LED auf Dauer nervt und es nicht ökologisch wertvoll ist eine Lampe dauerhaft leuchten zulassen, sollst du zum Abschluss einen Taster nutzen, um die LED ein- und auszuschalten.

DER ARDUINO UND DAS BREADBOARD



Dir ist mit Sicherheit direkt aufgefallen, dass dies kein originalgetreues Bild des Arduinos ist. Dabei handelt es sich lediglich um eine schematische Skizze, welche nur die allernötigsten Elemente beinhaltet. Diese Art des Arduinos wirst du im weiteren Verlauf des Workshops noch öfters sehen, immer dann wenn dir erklärt wird, wie du eine Schaltung aufbauen musst. Nachfolgend erfolgt eine kurze Erläuterung der wesentlichsten Bestandteile:

USB-Anschluss	Der USB-Anschluss bedient gleichzeitig zwei Funktionen. Zum einen wird darüber der Arduino mit Strom versorgt, sobald ihr diesen mit eurem Rechner verbunden habt. Zum anderen wird über die USB-Verbindung die Datenübertragung, sprich das Übertragen der Programme, realisiert.
Stromanschlüsse	Viele Bauteile, welche du im weiteren Verlauf des Workshops, aber auch schon in dieser Station, verwenden wirst, benötigen Strom. Um diese Bauteile mit Strom zu versorgen musst du die Verbindung gemäß nachfolgender Bezeichnung herstellen:  **Minuspol (-)**  **Pluspol (+)**
digitale Ein- und Ausgabepins	Digitale Signale können lediglich die Werte 1 und 0 beziehungsweise HIGH und LOW annehmen. Dabei stehen die Bezeichnungen prinzipiell für nichts anderes als für AN oder AUS. Diese Pins können zur Ausgabe, zum Beispiel zum Ein- und Ausschalten einer LED, sowie als Eingabe, beispielsweise für Schalter genutzt werden.
analoge Pins	Im Gegensatz zu digitalen Signalen können analoge Signale sehr viele Werte zwischen hohen und niedrigen Pegeln annehmen. Die genaue Anzahl der Werte - die Auflösung des digitalen Eingangs - liegt beim Arduino UNO bei 1024 Werten (10 Bit). Beim Arduino UNO entspricht der maximale Pegel 5V und der niedrige 0V. Diese individuellen Spannungswerte können mit den sechs analogen Pins (A0 - A5) des Arduino gemessen werden.
Steckbrett / Breadboard	Das Steckbrett ist neben dem Arduino selbst, das Gerät mit dem du im Folgenden am meisten zu tun haben wirst. Darauf sind sehr viele Löcher, mit denen mittels Kabel ein Schaltkreis realisiert werden kann. Das Besondere ist die Verbindung der Löcher untereinander, wie in der nächsten Abbildung dargestellt.

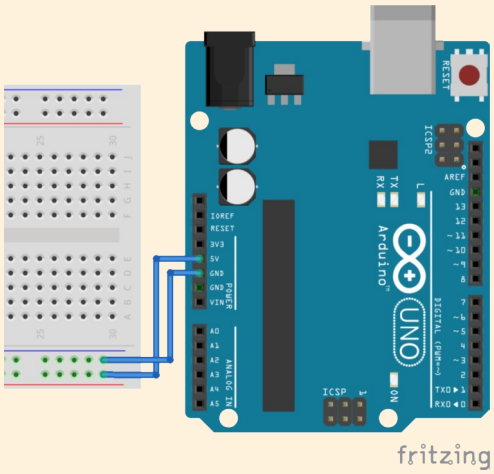


Beim Breadboard sind die äußeren Leisten durchgehend miteinander verbunden und die farbliche Gestaltung repräsentiert die Polung der Stromversorgung. Demnach bietet es sich an, den Pluspol an die rot markierte Leitung und die blau markierte, an den Minuspol anzuschließen. Die mittleren Anschlussblöcke sind immer vertikal, sprich immer fünf Löcher, miteinander verbunden. Keine Angst – diese Verbindungen siehst du nicht, da sich im Steckbrett unter der Plastikummantelung verbergen.

AUFGABE 1 – VERBINDEN DER ÄUSSEREN ANSCHLUSSLEISTEN

Um die Grundlage für die nachfolgenden Aufgaben zu schaffen, verbinden wir zunächst einmal die äußeren Anschlussleisten mit dem Plus- und Minuspol des Arduinos. Dies sollte dann wie in nebenstehender Abbildung aussehen.

Verwende dafür einfach zwei beliebige Kabel aus den zur Verfügung stehenden Materialien.



NUN LASSEN WIR UNS EIN LICHT AUFGEHEN

Soweit so gut, jetzt wollen wir natürlich einmal austesten, ob die Stromversorgung auch korrekt funktioniert. Zu diesem Zweck wollen wir eine LED zum leuchten bringen. Dazu brauchen wir noch nicht einmal programmieren, sondern realisieren einfach einen geschlossenen Stromkreis.

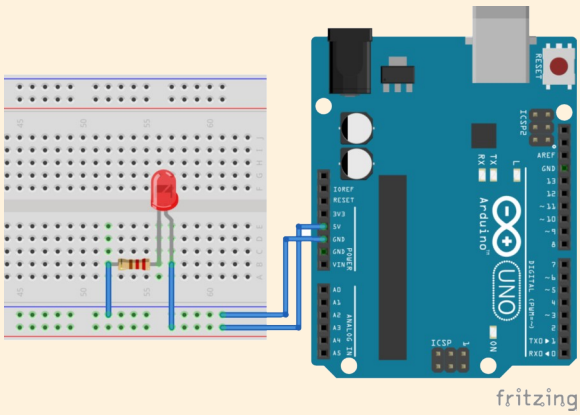
WHAT THE F@#K IST EINE LED?

LED, das 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. Welche Richtung das ist, könnt ihr bei LEDs ganz einfach erkennen: Ein Beinchen ist länger als das andere. Das lange muss mit dem Pluspol verbunden werden!

LEDs sind allerdings nicht sehr intelligent. Auch wenn sie nur sehr wenig Strom brauchen um zu funktionieren, nehmen sie jedoch alles an Strom auf, was der Stromkreis zu bieten hat. Dies bedeutet im schlimmsten Fall aber das Zerstören der LED – aber keine Angst, es gibt dafür ein Bauteil um dies zu verhindern. Dieses nennt sich **Widerstand** und muss bei Verwendung einer LED unbedingt im Stromkreis eingebaut werden – ob dies vor oder nach dem Bauteil geschieht spielt dabei keine Rolle.

AUFGABE 2 – BRINGE DIE LED IM STROMKREIS ZUM LEUCHTEN

Wir wissen nun, dass eine LED immer einen Widerstand benötigt. Schauen wir also nochmal auf unser Breadboard – die äußeren Anschlussleisten sind bereits verbunden. Fehlt nun also nur noch die LED und der Widerstand, zwischen Minuspol und kurzem Beinchen der LED!

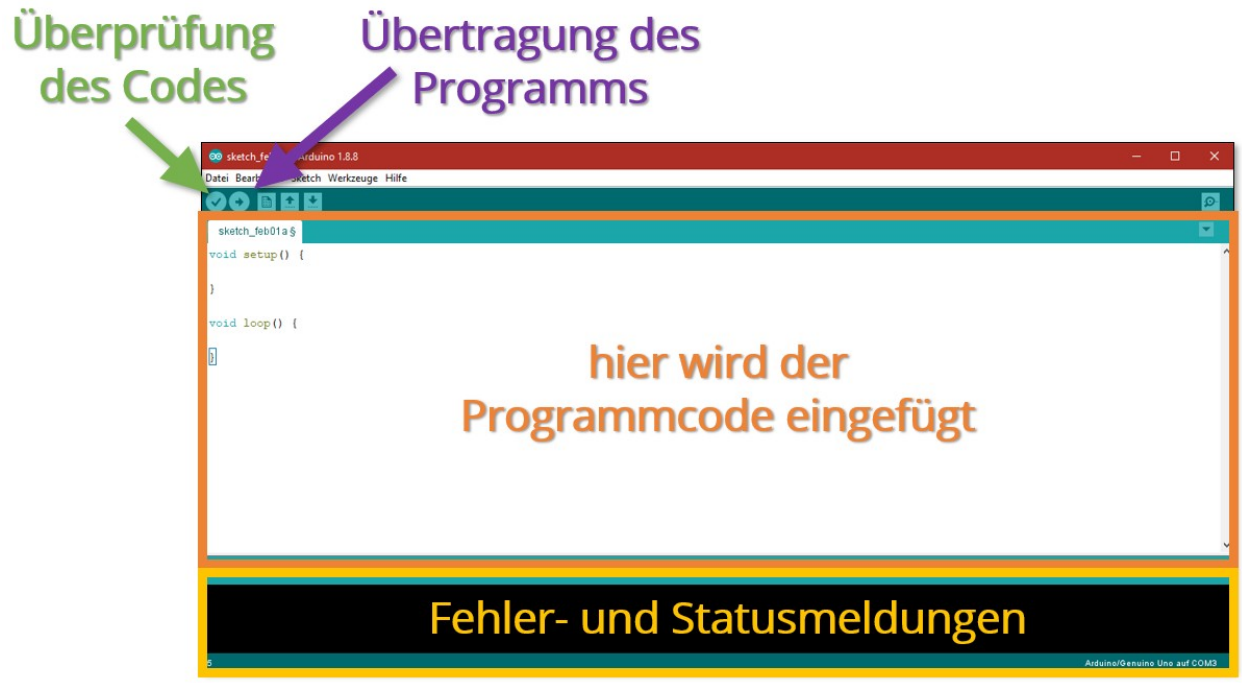


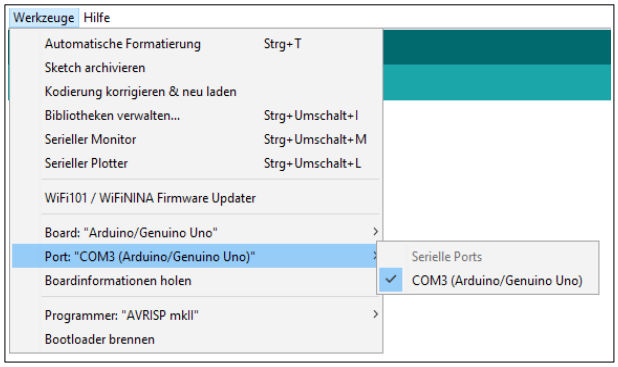
Alles verbunden? Dann verbinde jetzt das USB-Kabel mit dem Computer!

Die LED sollte jetzt leuchten! 👍😊

GRUNDLAGEN DER PROGRAMMIERUNG

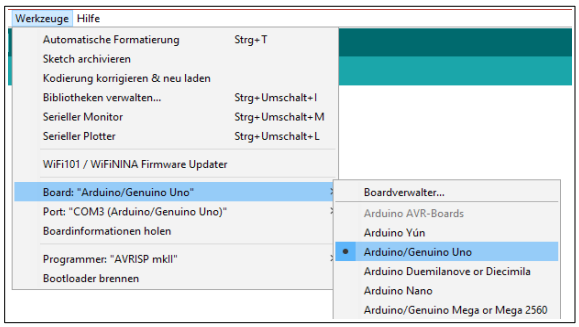
Programmiert wird mit der zum Arduino dazugehörigen Programmierumgebung. Schauen wir uns zunächst noch einmal die wichtigsten Elemente der Oberfläche an und verbinden das Gerät mit dem Rechner.





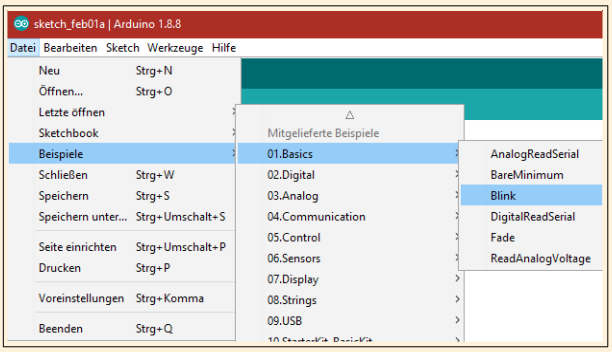
Das USB-Kabel hast du ja bereits an einem USB-Port am Rechner angeschlossen, allerdings weiß die Software noch nicht mit welchem Gerät und über welchen USB-Port sie kommunizieren soll. Dazu musst du zunächst unter `Werkzeuge > Port` (siehe linkes Bild) den richtigen Port auswählen.

Und unter `Werkzeuge > Board` (siehe rechtes Bild) überprüfen, ob der Mikrocontroller auch als Arduino Uno erkannt wurde.



AUFGABE 3 – TESTEN DER VERBINDUNG

Ohne unsere Schaltung zu verändern, können wir dennoch austesten ob die Verbindung zum Arduino funktioniert und ob wir erste Programme übertragen können. Navigiere dich dazu zunächst durch das Menü über `Datei > Beispiele > Basics > blink` und klicke dieses Beispiel an.



Die Übertragung auf den Arduino erfolgt mit dem  Button.

Damit wählst du ein Programm aus, welches die auf dem Arduino verbaute LED blinken lässt. Übertrage nun dein Programm und schau ob die LED wie gewünscht blinkt. Die `onboard`-LED findest du in der Nähe des 13. digitalen Pins!

Jetzt wird es Zeit, selbst ein Programm zu schreiben, dazu schauen wir uns erstmal an, wie ein Programm überhaupt aufgebaut ist. Wenn du einen neuen Sketch öffnest, so heißt ein Programm in der Programmierumgebung, ist die grundlegende Struktur bereits vorgegeben.

```
void setup() {
  ...
}

void loop() {
  ...
}
```

Der `setup()`-Programmcode wird immer nur ein einziges Mal ausgeführt, entweder wenn der Arduino das erste mal mit dem Strom verbunden oder der Reset-Knopf gedrückt wird. Wozu das wichtig ist und was man innerhalb dieses Programmabschnitts definieren sollte schauen wir uns nachfolgend an dieser Station noch an.

Im zweiten Teil hingegen werden Anweisungen in einer endlosen Schleife wiederholt, dies lässt sich auch aus dem Namen `loop()` erschließen, was auf deutsch übersetzt Schleife bedeutet. Beide Methoden sind zwingend notwendig, um das Programm erfolgreich kompilieren und ausführen zu können. "Kompilieren" bezeichnet die Übersetzung des Programms in einen Maschinencode, welcher vom Prozessor des Arduinos verstanden werden kann - dies übernimmt aber zum Glück die Programmierumgebung für uns.

PROGRAMMIERUMGEBUNG / ENTWICKLUNGSUMGEBUNG

Du hast jetzt schon ein paar mal den Begriff „Programmierumgebung“ gelesen. Eine gleichwertige Bezeichnung dafür wäre auch Entwicklungsumgebung und auf englisch IDE. Dies steht konkret für Integrated Development Environment, beziehungsweise integrierte Entwicklungsumgebung.

Diese Softwarelösungen unterstützen dabei Programmiererinnen und Programmierer aktiv beim Entwicklungsprozess. Dies kann zum Beispiel durch sogenanntes Syntax Highlighting geschehen, bei dem Schlüsselwörter farblich markiert und hervorgehoben werden oder auch durch die Integration von Statusmeldungen, welche einem ein direktes Feedback zur Korrektheit des Codes oder zu Übertragungsproblemen geben.

JETZT WIRD PROGRAMMIERT – LASSE EINE LED BLINKEN

Ähnlich zum vorherigen Beispiel, bei dem wir die LED des Arduinos zum Blinken gebracht haben, wollen wir jetzt die auf dem Breadboard platzierte LED zum Blinken bringen.

Was brauchen wir dazu alles? Als erstes müssen wir die Schaltung insofern abändern, dass der Pluspol, sprich das lange Beinchen der LED, nicht mehr mit der roten Anschlussleiste auf dem Breadboard verbunden ist. Um die Steuerung mit dem Arduino zu realisieren, ändern wir die Verbindung so um, dass das Kabel in einen der digitalen Pins gesteckt wird.

Im nächsten Schritt definieren wir uns im Programmcode eine Variable mit Hilfe derer wir uns merken, an welchem Pin wir die LED angeschlossen haben. Dies machen wir in einem derzeit noch unbekannten dritten Abschnitt des Sketches, die Initialisierung. Dies geschieht noch vor dem `setup()`-Bereich und dient der Definition von Variablen und weiterführend auch von sogenannten Objekten. Eine Variable ist dabei ein Speicherbereich, quasi eine Art Eimer, in dem ihr Daten ablegen könnt.

Beispielhaft sieht das dann wie folgt aus:

```
int ledRot = 6;
```

Was bedeutet das? Das `int` steht für den Datentyp Integer, welcher Ganzzahlen, sprich `1,2,3,...` beinhaltet. Dazu gleich mehr auf der nächsten Seite.

Die nächste Bezeichnung `ledRot` ist eine Variablenbezeichnung. Unter diesem Namen können wir nachfolgend auf die Variable zugreifen, den Wert auslesen oder ihr einen neuen Wert vergeben.

Die Bezeichnung kann völlig frei gewählt werden, sollte aber prinzipiell sinnig zu dem sein, was sie repräsentiert. Der letzte Teil der oberen Zeile steht für den Wert, welchen wir der Variable zuweisen. Im konkreten Beispiel steht die Zahl 6 für den Pin, welchen wir nutzen wollen um die LED anzusteuern.

DATENTYPEN

Boolean
Übersetzt bedeutet das Wahrheitswert, welche entweder TRUE (wahr) oder FALSE (falsch) sein kann, beziehungsweise in der Sprache eines Computers 1 oder 0.

INTEGER
Speichert Ganzzahlen ohne gebrochenen Dezimalanteil. Bis auf die Division kann unter alleiniger Verwendung von Ganzzahlen ohne Probleme dieser Datentyp verwendet werden.

FLOAT
Eine Fließkommazahl speichert ebenfalls den gebrochenen Anteil einer Zahl. Damit können sämtliche Rechenoperationen ausgeführt werden, ohne Sorge dass ein etwaiger gebrochener Anteil abgeschnitten wird.

CHAR
Der Datentyp wird benutzt um einzelne Zeichen zu speichern. Dabei steht Char für Charakter, sprich ein einzelnes Zeichen. Mann erkennt einen Charakter daran, dass er in einfache Anführungsstriche gesetzt wird, z.B. 'A'.

STRING
Sogenannte Strings sind Zeichenkettenfolgen, welche beliebig viele aneinander gekettete Buchstaben beinhalten. Strings werden in Anführungszeichen gesetzt, Beispiele für Strings sind "Informatik", "Hallo Max!", "@%#&$", "xyz" sowie auch " ".

In einem nächsten Schritt müssen wir dem Arduino noch mitteilen, dass unser nun belegter Pin zur Ausgabe genutzt werden soll. Dafür nutzen wir die Funktion `pinMode(PIN, OUTPUT)` und legen somit fest, dass an dem Pin ein Gerät ist, was wir zur Ausgabe nutzen möchten. Dies wird innerhalb des `setup()`-Methode definiert.

AUFGABE 4 – VARIABLE UND OUTPUT FESTLEGEN

Verbinde deine LED mit einem beliebigen digitalen Pin. Initialisiere danach wie beschrieben eine Variable für die LED und lege den belegten Pin als Output (Ausgang) fest. Beachte dabei, dass jeder Befehl mit einem Semikolon ; beendet wird.

Jetzt wollen wir die LED zum Leben erwecken. Die zweite Funktion die wir dazu nutzen ist `digitalWrite(PIN, HIGH)` oder `digitalWrite(PIN, LOW)` Wichtig zu wissen ist, dass digitale Signale nur zwei Arten von Werten kennen. Spezifischer wäre dies `HIGH (1)` für das Anschalten der Stromversorgung oder `LOW (0)` für das Ausschalten. Da unsere LED blinken soll, wollen wir zunächst die LED anschalten und direkt danach wieder ausschalten.

AUFGABE 5 – STEUERN DER LED

Füge die korrekte Funktion zur Steuerung der Stromversorgung der LED hinzu. Dabei soll zunächst die LED ein- und direkt danach wieder ausgeschalten werden. Nutze zur Festlegung des Pins deine zuvor festgelegte Variable. Das wird mehrmals wiederholt, muss also in der `loop()`-Methode stehen. Übertrage nun dein Programm auf den Arduino und beobachte das Verhalten der LED. Was stellst du fest?

Richtig. Irgendwie leuchtet die LED gar nicht, obwohl du im Programm den Strompegel auf HIGH gesetzt hast und das Statusfenster der Programmierumgebung keinen Fehler geliefert hat. Woran könnte das liegen? Durch die iterative Abarbeitung des Programmcodes, also der Abarbeitung Zeile für Zeile, wird direkt nach dem Setzen des Strompegels auf HIGH dieser sofort wieder rückgängig gemacht. Wir Menschen bekommen von den Verarbeitungsschritten überhaupt gar nichts mit, da der Prozessor des Arduinos die Befehle deutlich schneller abarbeitet als wir es nachvollziehen können.

Zur Lösung des Problems lernen wir jetzt eine weitere Funktion kennen, welche `delay(Zeit in ms)` heißt. Wir verzögern damit die Ausführung des Programms um eine bestimmte Zeit und erlauben der LED damit auch wirklich zu leuchten. Stelle dir dies vereinfacht wie ein Stoppschild im Straßenverkehr vor – wir halten den Verkehrsfluss kurz an und warten eine vorgeschriebene Zeit ab, bis weiter gefahren werden kann.

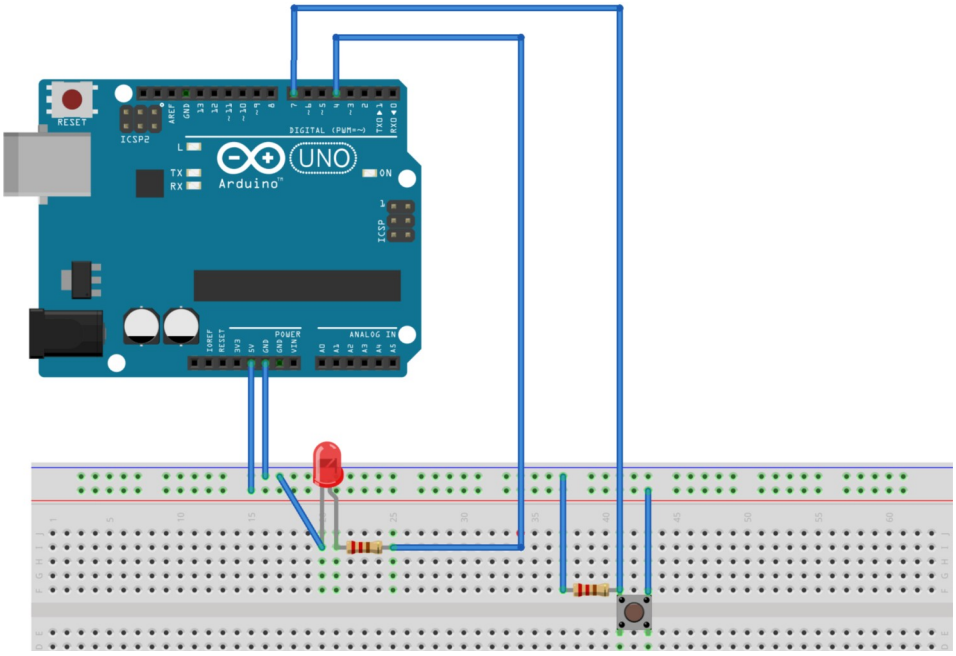
AUFGABE 6 – FÜGE EINE VERZÖGERUNG HINZU

Ergänze dein bis jetzt vorhandenes Programm um die eben benannte Verzögerungsfunktion. Überlege dir dazu, wo die Pausenfunktion platziert werden muss. Übertrage anschließend das Programm wieder auf den Arduino und schau dir nun an, ob du die LED leuchten siehst.

Dir geht es immer noch zu schnell? Dann variiere die Zeit innerhalb der Funktion. Bedenke dabei: 1000 ms entsprechen 1 Sekunde.

ICH KLICK MIR DIE WELT WIE SIE MIR GEFÄLLT – TASTER ZUR STEUERUNG

Sehr gut, dein erstes Programm funktioniert schon mal richtig gut. Also schauen wir uns nun an, wie wir das Projekt trotzdem weiter ergänzen und verbessern. Zu diesem Zweck wollen wir nun einen sogenannten Taster nutzen. Dazu müssen wir als erstes die Schaltung etwas verändern.

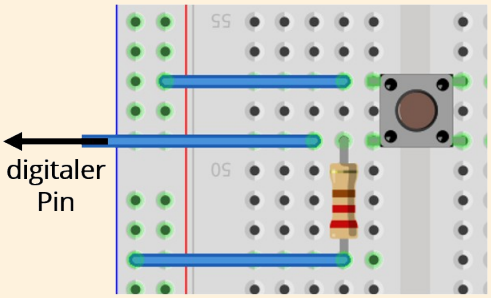


Wenn du wissen willst wie und warum diese Schaltung funktioniert, lies dir doch einfach den nächsten Hinweis durch. Der Taster sieht dabei wie in der Abbildung auf der rechten Seite aus. Zusätzlich zu dem Gehäuse hat der Taster auch noch eine gelbe Kappe, um ihn besser drücken zu können.

AUFGABE 7 – REALISIERE DIE GEZEIGTE SCHALTUNG

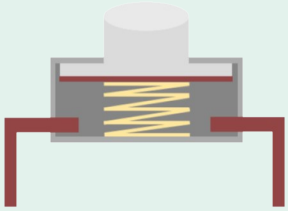
Realisiere nun die Schaltung mit dem neuen Bauelement, dem Taster, wie es in der obigen Schaltung gezeigt ist.

Achte auf die Polaritäten der Kabel:
Die rechte Seite des Widerstandes soll mit dem Minuspol verbunden werden. Die rechte Tasterseite und der Widerstand werden gemeinsam mit einem digitalen Pin verbunden. Die andere Seite des Tasters wird mit dem Pluspol verbunden.



DER STROMFLUSS IN EINER TASTERSCHALTUN

Ein Taster, das ist ein Bauelement, welches solange den Strom leitet, wie der Knopf nach unten gedrückt ist. Das Innere eines Taster sieht wie in der Abbildung rechts aus, wobei alle braunen Teile elektrische Leiter darstellen.



Was passiert wenn der Schalter nach unten gedrückt wird?

_____

Werfen wir nochmal einen Blick auf den derzeitigen Aufbau unserer Schaltung. Über einen Widerstand sind die äußeren Enden mit dem Plus- und Minuspol verbunden. Das mittlere Kabel führt zu einem digitalen Pin, wie wir ihn bei der LED auch schon benutzt haben. Bei einem nicht gedrückten Taster ist über den Widerstand nur eine Verbindung zum Minuspol realisiert, am digitalen Pin liegt damit `LOW (GND)` an.

Durch das Drücken des Tasters nach unten, erreichen wir auch auf der rechten Seite einen geschlossenen Stromkreis, dass heißt auch durch das Bauteil fließt ein Strom. Was passiert nun? Jetzt spielt der Widerstand seine bedeutsame Rolle in der Schaltung aus, denn vereinfacht kannst du dir merken, dass sich Strom immer den Weg des geringsten Widerstandes sucht. Dadurch realisieren wir eine Verbindung zum Pluspol, da der Fluss durch den Widerstand zu aufwendig wäre. Am digitalen Pin liegt jetzt `HIGH (+5V)` an.

WAS SOLL PASSIEREN WENN DER TASTER GEDRÜCKT WIRD?

Überlege dir zunächst was passieren soll wenn der Taster gedrückt oder nicht gedrückt ist. Trage unten deine Vorstellungen ein!

	Taster gedrückt	**Taster nicht gedrückt**
Verhalten der LED		

Wir müssen es also im Programmcode hinbekommen, irgendwie eine Entscheidung zu fällen, in Abhängigkeit ob der Taster gedrückt wurde. Dazu lernen wir nun eine neue algorithmische Grundstruktur, die sogenannte einfache Verzweigung kennen.

Vereinfacht ausgedrückt kann man diese Struktur wie folgt erklären:

WENN (Taster gedrückt) **DANN** LED _____ **SONST** _____

Trage auf den Strichen das Verhalten der LED ein – also **AN** oder **AUS**!

Im Programmcode wird dies dann wie rechtsstehend umgesetzt. Innerhalb der Klammer überprüfen wir unsere Bedingung, sprich ob an unserem Pin (das mittlere Kabel unserer Tastenschaltung) `HIGH` oder `LOW` anliegt. Dazu nutzen wir die Funktion `digitalRead(PIN)`. Wir lesen also aus, was an unserem Pin anliegt, wobei die Funktion `HIGH` als Zahlenwert 1 und `LOW` als Zahlenwert 0 ausgibt.

```
if(BEDINGUNG){
      LED _______
} else {
      LED _______
}
```

Wir haben ja bereits geklärt, das beim Drücken des Taster `HIGH` am Pin anliegt, damit können wir unsere Bedingung wie folgt formulieren:

```
if(digitalRead(PIN) == HIGH) { … } else { … }
```

Was fällt hierbei direkt auf? Richtig – wir verwenden ein doppeltes Gleichheitszeichen. Dieses gehört zur Gruppe der Vergleichsoperatoren, welche es uns ermöglichen Werte miteinander zu vergleichen.

VERGLEICHSOPERATOREN

Vergleichsoperatoren vergleichen zwei Werte miteinander. Je nach Vergleich, kann diese Überprüfung richtig oder falsch sein. Das ist auch genau das, was beim Programmieren innerhalb der Verzweigungsbedingung zurückgegeben wird, spezifischer `TRUE` oder `FALSE`.

x == y	(x ist gleich y)	**x != y**	(x ist nicht gleich y)
x < y	(x ist kleiner als y)	**x > y**	(x ist größer als y)
x <= y	(x ist kleiner als oder gleich zu y)		
x >= y	(x ist größer als oder gleich zu y)		

AUFGABE 8 – TASTER-STEUERUNG ERGÄNZEN

Die Grundlagen kennst du jetzt – du musst sie nur noch zu einem großen Ganzen zusammenfügen. Als erstes beginnst du damit, für den Pin, an dem der Taster angeschlossen ist, eine Variable zu definieren.

Danach ergänzt du im `setup()`-Bereich die `pinMode()`-Funktion für diesen Pin. Bedenke dabei: Wir haben jetzt keinen OUTPUT, sondern wollen etwas einlesen – also einen INPUT.

Im `loop()`-Bereich folgt nun unsere einfache Verzweigung, mit der Bedingung und dem in deiner Tabelle festgehaltenen Verhalten der Led.

DAS ENDERGEBNIS

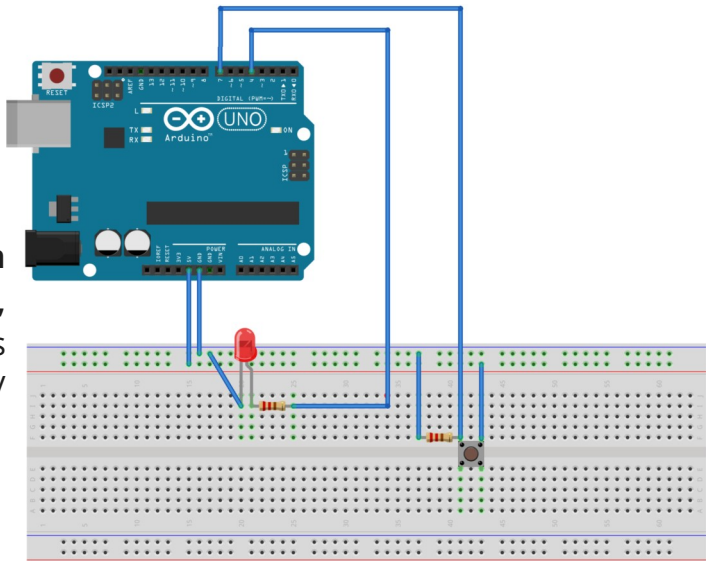
Dein Programmcode sollte ungefähr so aussehen. Alles nach den zwei **Schrägstrichen** sind Kommentare, die dir nochmals die Bedeutung erklären.

```
int LED = 4;
//an diesem Pin ist die LED angeschlossen
int taster = 7;
//an diesem Pin ist der Taster angeschlossen
int tasterstatus = 0;
//der Status des Tasters wird mit 0 (low) initialisiert,
sprich "nicht gedrückt"

void setup(){
     pinMode(LED, OUTPUT);
     //konfiguriert den LED - Pin als Ausgabe(pin)
     pinMode(taster, INPUT);
     //konfiguriert den Taster - Pin als Eingabe(pin)
}

void loop(){
     tasterstatus = digitalRead(taster);
     //Einlesen des Tasterstatus
     if (tasterstatus == HIGH){
          //wenn Taster gedrückt, dann ...
          digitalWrite(LED, HIGH);
          //die Spannung an diesem wird HIGH gesetzt, dass
          bedeutet auf 5V
     } else { // ... sonst ...
          digitalWrite(LED, LOW);
          //die Spannung an diesem wird LOW gesetzt, dass
          bedeutet auf 0V
     }
}
```

Und deine **Schaltung, mit dem Arduino und dem Breadboard,** sollte aussehen, wie in der rechts stehende schematischen Skizze / Zeichnung.



Grafik auf dem Deckblatt: *A Arduino Uno board, JotaCartas, CC BY-SA 2.0, https://creativecommons.org/licenses/by/2.0/deed.en, https://commons.wikimedia.org/wiki/File:Arduino-uno-perspective-transparent.png*
Screenshots: *fritzing electronics made by easy und Arduino IDE 1.8.12 (windows)*
Alle weiteren Grafiken: *Patrick Binkert, EduInf@TUD*