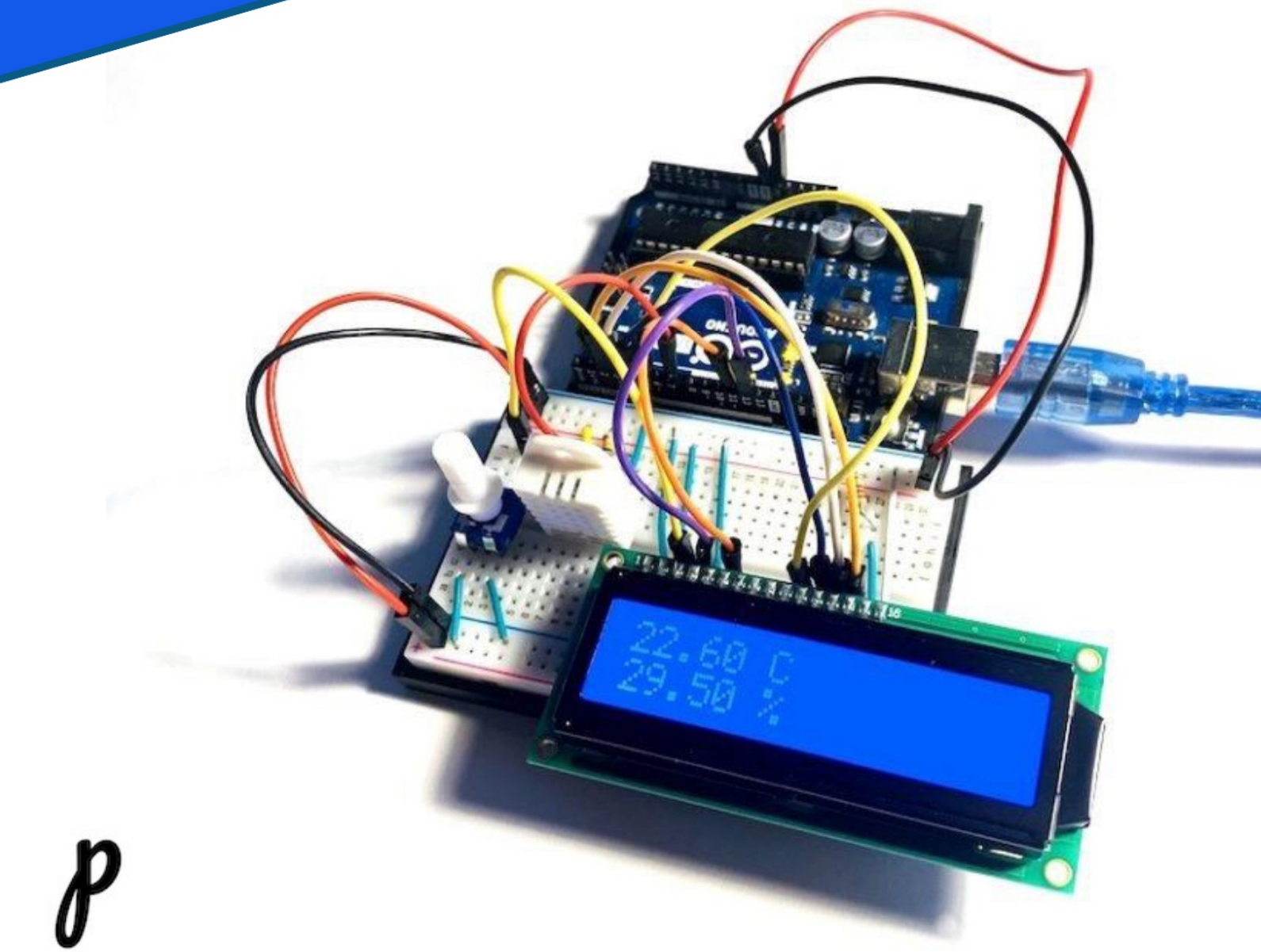
Vom Anfänger zum Maker

MIT ARDUINO & ESP8266



VOM ANFÄNGER ZUM MAKER – MIT ARDUINO & ESP8266

VORWORT

Vielen Dank dafür, dass du dich für dieses E-Book entschieden hast.

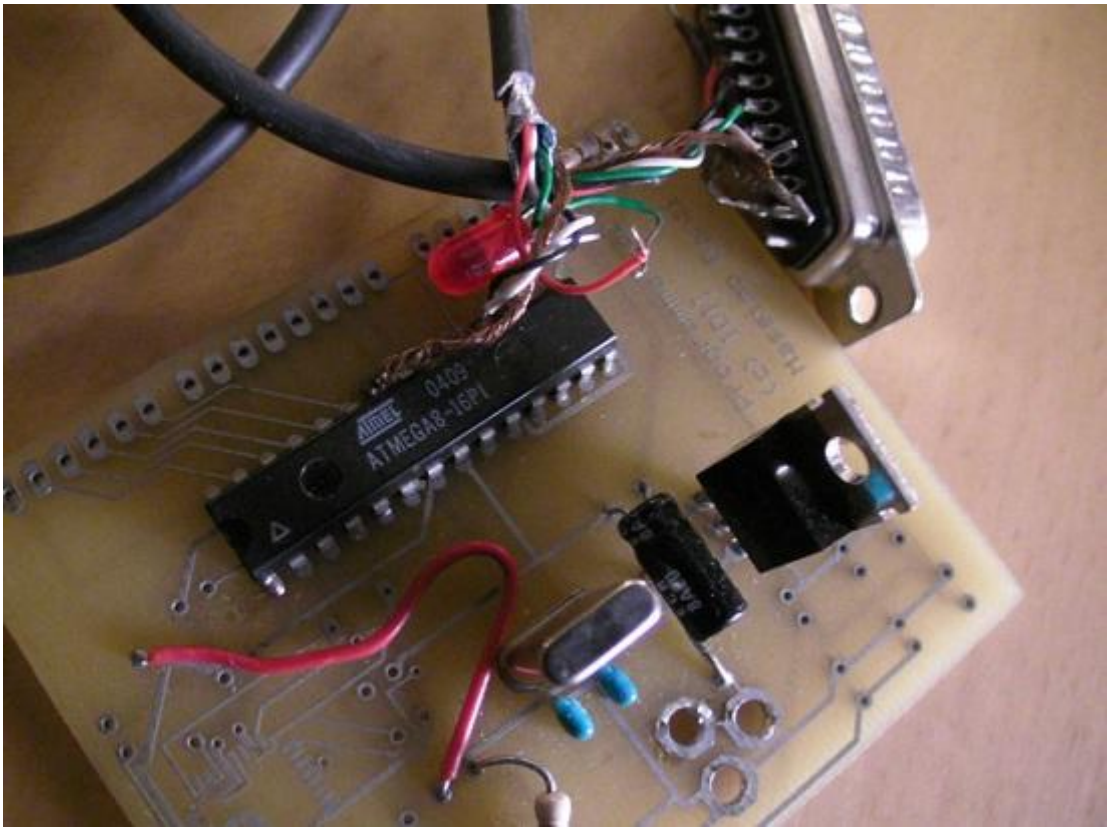
Wir möchten dich gar nicht lange mit einem Vorwort aufhalten, sondern dir gleich viel Spaß auf deiner Reise durch die Welt des Arduinos wünschen. Solltest du an einer Stelle dieses E-Books oder bei einem Projekt nicht weiterkommen, schreibe uns gerne eine E-Mail an info@polluxlabs.net

Und nun, viel Spaß mit deinem Arduino!

1 DER ARDUINO UNO

Der erste Arduino Microcontroller ist auch gleichzeitig der bekannteste: Arduino UNO. Seine Entwicklung begann im Jahr 2003 in Italien. Einige Studenten haben ihn hier zusammen mit dem Ziel entwickelt, einen günstigen und leicht programmierbaren Microcontroller für Bastler und Studierende zu entwickeln.

Und das mit Erfolg. Im Jahr 2005 erblickte der Arduino UNO das Licht der Welt – und es dauerte nicht lange, bis er die Herzen der Maker rund um den Globus erobert hatte.



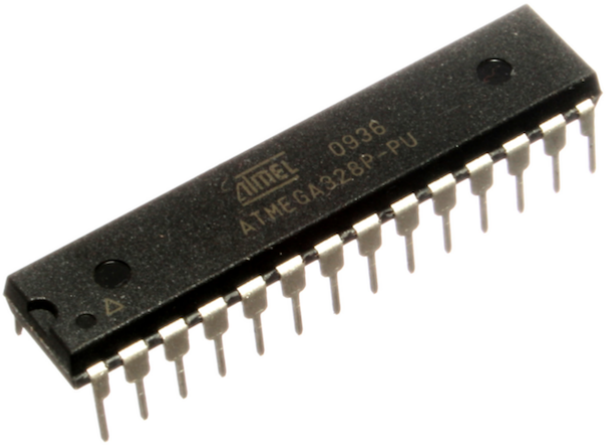
Der erste Arduino-Prototyp, Foto: Phillipptorrone, Wikipedia

Auch heute ist es für viele immer noch der erste Microcontroller, den sie in Händen halten. Er ist erschwinglich im Original und noch erschwinglicher in einer Kopie. Außerdem bietet er Komfort beim Aufbau von Projekten und

Programmieren. **Lass uns zusammen einen Blick auf seine wichtigsten Features und Komponenten werfen.**

DAS GEHIRN: ATMEGA328P

Kein Microcontroller ohne Microchip. Beim Arduino UNO ist dies der ATmega328P, der zentral in einem Sockel auf dem Arduino Board sitzt.



Microchip ATmega328P

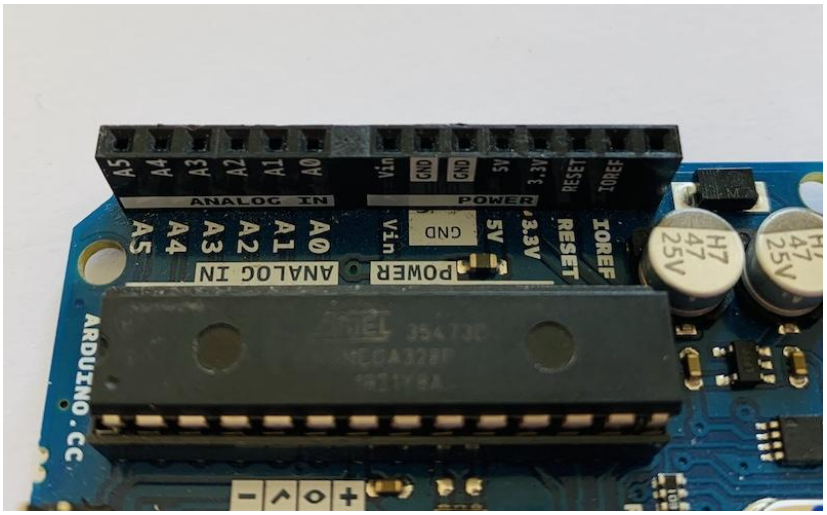
Auf diesem Chip landen deine Programme, die du hochlädst – wofür dir 32KB zur Verfügung stehen. **Gemessen an modernen Speichermedien klingt das nach nichts, du wirst aber sehen, dass du damit schon unheimlich viel anfangen kannst.**

Wenn du dir den ATmega328P genauer anschaust, siehst du, dass er 28 “Beinchen” – sogenannte Pins – hat. **Von diesen Pins kannst du 23 programmieren.** Wenn du deinen Arduino UNO umdrehst, erkennst du die Leiterbahnen, die unter anderem zu den Buchsen an den Seiten des Boards führen.

Wenn du hier also zum Beispiel eine LED in die Buchse A5 steckst, verbindest du sie dadurch mit dem entsprechenden Pin am Microchip.

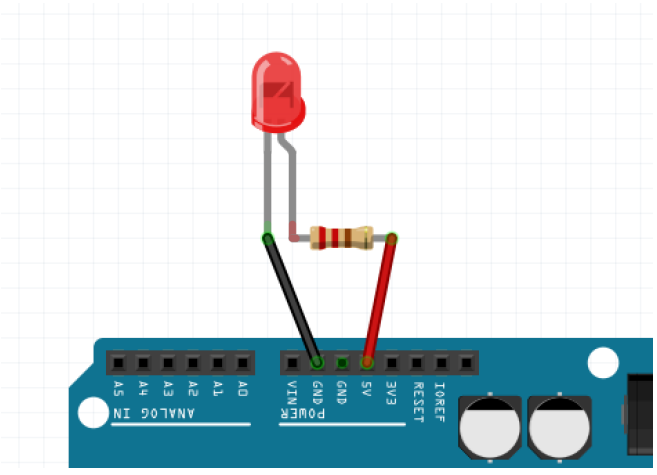
DIE WICHTIGSTEN PINS DES ARDUINO UNO

Als nächstes werfen wir einen Blick auf die Pins, die du im später benötigen wirst.



Power- und Analog-Pins

Hier haben wir zunächst Pins, die mit **Power** gekennzeichnet sind. Die beiden Pins **GND** dienen als “Minuspol” oder “Erde”. Hier kannst du beispielsweise die Kathode (Minus, kurzes Bein) einer LED anschließen, um sie zum Leuchten zu bringen.



Eine LED am Arduino UNO

In der Skizze oben siehst du, dass die Anode (Plus, langes Bein) der LED über einen Vorwiderstand an **5 Volt** angeschlossen ist. Ihr Minuspol führt direkt zu **GND**. Über LEDs und Widerstände lernst du später mehr.

Das führt uns direkt zu den beiden Pins **5V** und **3V3**. Hierüber kannst du – wie die LED oben – Bauteile mit Strom versorgen. Je nachdem, wie viel Spannung (Volt) sie benötigen, stehen dir hierfür entweder 5 Volt oder 3,3 Volt zur Verfügung. **Die meisten Hobby-Bauteile wie Servo-Motoren, Sensoren etc. benötigen eine dieser beiden Spannungen.**

Fehlt nur noch der Pin **VIN**. Hierüber kannst du den Arduino UNO mit Strom versorgen – das schauen wir uns aber in Kürze genauer an, wenn wir über die Stromversorgung sprechen.

DIE ANALOG-PINS

Zunächst betrachten wir die **Analog-Pins A0 bis A5**. Mit diesen sechs Pins kannst du Signale “messen”, die sich kontinuierlich verändern können. Tatsächlich sind das sich verändernde Spannungen zwischen 0 und 5 Volt. Ein Beispiel ist ein Temperatursensor, der sein Ausgangssignal (also die Spannung) verändert, wenn es wärmer oder kälter wird.

Diese Spannung wird im Arduino von einem sogenannten **Analog-Digital-Konverter (ADC)** in eine entsprechende Zahl von 0 bis 1023 umgewandelt.

Die Analog-Pins sind also immer dann nützlich, wenn du ein veränderbares Signal messen möchtest – und nicht nur wie bei digitalen Signalen entweder eine Null oder eine Eins. Allerdings kannst du über die Analog-Pins kein veränderbares Signal ausgeben, das geht nur über einige der Digital-Pins – was uns direkt auf die andere Seite des Arduino UNO führt.

DIE DIGITAL-PINS

Hier findest du insgesamt 13 Digital-Pins. Allerdings stehen dir für deine Projekte nur die Pins 2 bis 13 zur Verfügung – Pin 0 und 1 sind für etwas anderes reserviert und können nicht programmiert werden.

Mit den Digital-Pins kannst du die Signale Null (Aus) und Eins (An) messen und ausgeben. Das bedeutet zum Beispiel, dass du eine LED über einen der Pins an- und ausschalten kannst. Oder messen kannst, ob eine Taste gedrückt wurde.

Einige der Digital-Pins sind mit einer Tilde (~) gekennzeichnet. Über diese Pins kannst du das erreichen, was du vielleicht eher von den Analog-Pins erwartest hättest: Du kannst hierüber ein analoges, also sich veränderndes Signal ausgeben. **Damit kannst du zum Beispiel eine LED mit unterschiedlicher Helligkeit leuchten lassen.**

Der Fachbegriff hierfür heißt Pulsweitenmodulation (PWM). Hier erfährst du mehr zu diesem Thema.

STROMVERSORGUNG

Um seine Arbeit aufnehmen zu können, benötigt dein Arduino UNO vor allem eines: Strom. Hierfür stehen dir mehrere Möglichkeiten zur Verfügung. Zunächst die USB-Buchse – diese wirst du sicherlich am häufigsten verwenden, da du hierüber auch deine Programme auf den Arduino hochlädst. **Der Arduino erhält seinen Strom dann von deinem PC oder Laptop.**

Neben der USB-Buchse befindet sich noch ein weiterer Anschluss. Hierüber kannst du einen 9V-Block anschließen. Damit machst du deinen Arduino unabhängig von einem Computer und kannst ihn transportieren.

Eine weitere Möglichkeit ist der Pin **VIN**, den wir oben schon kurz angeschaut haben. Auch hierüber kannst du Batterien anschließen. Ihren Pluspol verbindest du mit **VIN**, den Minuspol mit **GND**.

DEN ARDUINO NEU STARTEN

Zuletzt werfen wir noch einen Blick auf einen Button, der sich auf deinem Arduino UNO neben der USB-Buchse befindet: den Reset-Knopf. Wenn du deinen Arduino neu starten möchtest, genügt ein Drücken dieses Buttons und das Programm in seinem Speicher beginnt von vorne.

Und das soll es vorerst gewesen sein. Du kennst nun die wichtigsten Komponenten deines Arduino UNO, die du auch im weiteren Verlauf verwenden wirst.

2 DIE ARDUINO IDE

Für Einsteiger ist die Arduino IDE (Integrated Development Environment) meist die erste Wahl – und das zu Recht. Du kannst zahlreiche Microcontroller mit ihr programmieren sowie Bibliotheken für Sensoren, Displays etc. verwalten. Außerdem besitzt sie den “Seriellen Monitor”, in dem du Daten ausgeben und auf Fehlersuche gehen kannst.

Einsteigerfreundlich ist auch die Tatsache, dass die Arduino IDE vieles verbirgt, was für Anfänger nicht relevant ist. So fällt der Einstieg in das eigentlich recht komplexe Thema Hardware-Programmierung leicht.

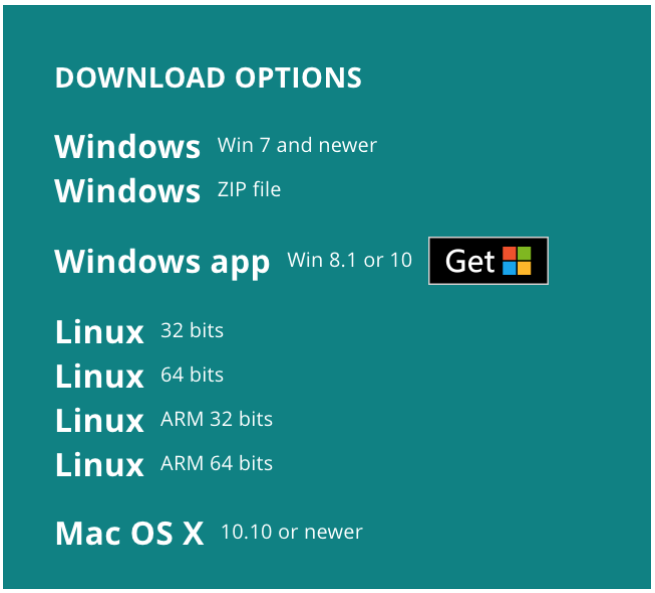
In den folgenden Abschnitten schauen wir uns an, wie du die Arduino IDE installierst. Anschließend folgt eine kleine Tour durch die wichtigsten Funktionen, um mit den ersten Projekten starten zu können.

Hinweis: Die Arduino IDE befindet sich aktuell (Herbst 2022) im Übergang von der Version 1.8.x zur Version 2.0 – die neue Version hat den Beta-Status zwar verlassen und ist veröffentlicht, viele Maker verwenden allerdings noch die Version 1.8.x. Deshalb findest du hier zunächst die Einführung zur “alten” IDE und anschließend mehr darüber, was sich nach dem Update geändert bzw. was in der neuen Version hinzugekommen ist.

INSTALLATION

Die Arduino IDE gibt es für Windows, macOS und Linux und ist mit ein paar Klicks auf deinem Rechner installiert.

Auf der offiziellen Download-Seite auf arduino.cc findest du die jeweils aktuelle Version. Ein Klick auf dein Betriebssystem startet den Download. Wenn du die Version 1.8.x installieren möchtest, scrolle etwas nach unten bis zum Abschnitt **Legacy IDE (1.8.X).**



Sobald die Datei auf deinem Rechner ist, starte sie (bzw. öffne die ZIP-Datei) und folge den weiteren Anweisungen. Anschließend kannst du die Arduino IDE öffnen.

ALTERNATIVEN

Ebenso offiziell ist auch der **Arduino Web Editor**. Hiermit kannst du deinen Arduino direkt im Browser programmieren. Ganz ohne Download geht das allerdings auch nicht, denn du benötigst das Plugin **Arduino Create Agent** für die Kommunikation zwischen Rechner und Arduino. Außerdem musst du ein kostenloses Benutzerkonto anlegen, um den Editor verwenden zu können.

Erfahrene Nutzer können auch die **Beta-Version** und **Nightly Builds** verwenden. Hierbei handelt es sich um Versionen, die bereits künftige Features enthalten. Allerdings können hier noch Bugs vorhanden sein, weswegen Einsteiger sie eher nicht verwenden sollten.

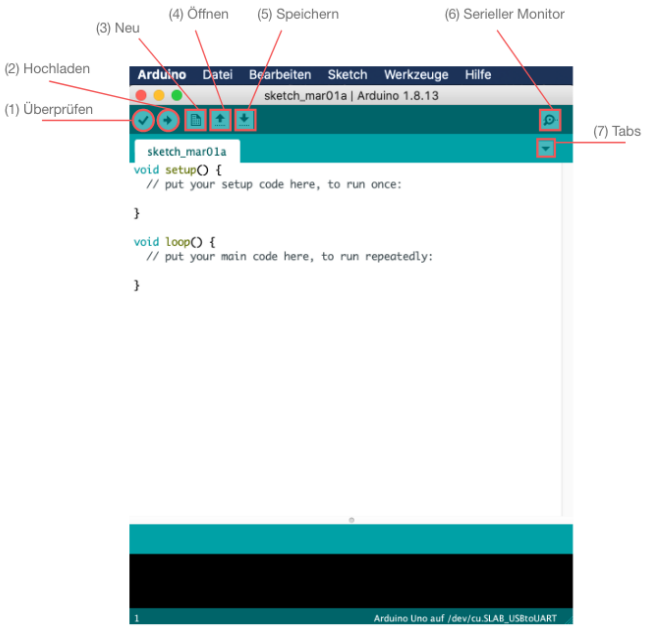
LERNE DIE IDE KENNEN

Wenn du die Arduino IDE zum allerersten Mal startest, passiert nichts außergewöhnliches. Es gibt keinen Willkommens-Screen, kein Tutorial und keine Tour durch das Programm. Deshalb übernehmen wir das jetzt.

In dieser und den folgenden Lektionen arbeiten wir mit der Mac-Version der Arduino IDE. **Das Programm ist jedoch auf anderen Betriebssystemen weitestgehend identisch.**

DAS CODE-FENSTER

Nach dem Programmstart öffnet sich ein Fenster für den Code – **in der Arduino-Welt auch Sketch genannt.** Diesen schauen wir uns später genauer an, jetzt werfen wir zunächst einen Blick auf die Buttons, die sich im oberen Bereich des Fensters befinden. Hier findest du wichtige Funktionen, die du beim Programmieren deines Arduinos immer wieder benötigst.



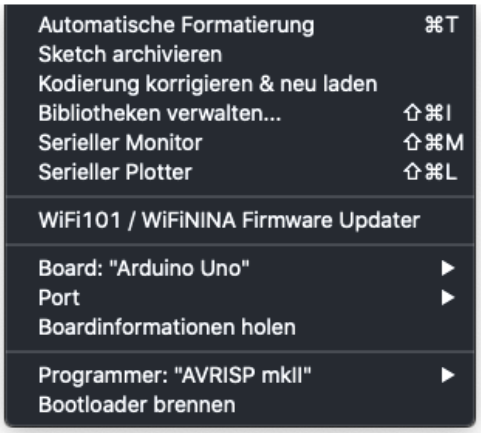
Von links nach rechts sind das:

1. **Überprüfen:** Mit einem Klick auf diesen Button überprüfst du deinen Code auf Fehler. Findet die Arduino IDE einen Fehler, wird die entsprechende Zeile rot markiert und im unteren Bereich des Fensters die Fehlermeldung und oft auch ein Hinweis eingeblendet.
2. **Hochladen:** Mit diesem Button lädst du deinen Sketch auf den Arduino hoch. Vor dem Upload findet auch immer zunächst eine Überprüfung des Codes statt.
3. **Neu:** Dieser Button öffnet ein neues Fenster.
4. **Öffnen:** Hiermit kannst du einen gespeicherten Sketch öffnen.
5. **Speichern:** Richtig, mit diesem Button kannst du deinen Sketch auf einem Datenträger speichern.
6. **Serieller Monitor:** Dieser Button öffnet den Seriellen Monitor, sofern du deinen Arduino am Rechner angeschlossen hast. Mit dem Seriellen Monitor beschäftigen wir uns später im Kurs genauer.
7. **Tabs:** Du benötigst nicht für jeden Sketch auch ein neues Fenster, sondern kannst diese auch nebeneinander in einem Fenster öffnen – so wie du es von deinem Browser kennst. Mit diesem Button kannst du deine Tabs organisieren.

DIE MENÜLEISTE

Neben den Funktionen innerhalb des Fensters für deinen Code gibt es oben auch eine Menüleiste. Hier findest du ebenfalls die oben aufgeführten Funktionen – aber auch noch einiges mehr.

Zunächst werfen wir allerdings einen Blick in das Menü **Werkzeuge**. Hier findest du neben **Board:** den Microcontroller, der gerade in der Arduino IDE verwendet wird.



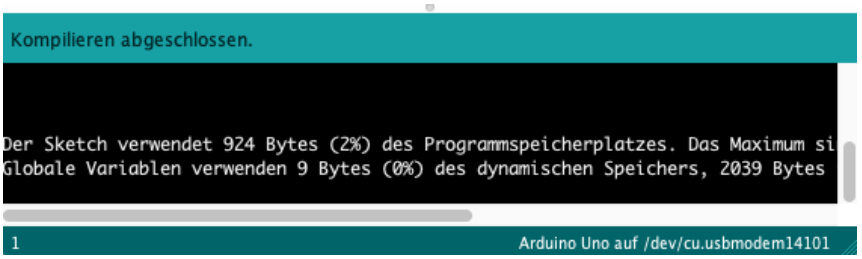
Wenn bei dir hier noch nicht **Arduino Uno** steht, klicke auf den Menüpunkt und wähle zunächst **Arduino AVR Boards**. Jetzt öffnet sich ein weiteres Untermenü, in dem du dann den **Arduino Uno** auswählen kannst.

Hast du deinen Arduino bereits mit deinem Rechner per USB verbunden? Wenn nicht, hole das jetzt nach.

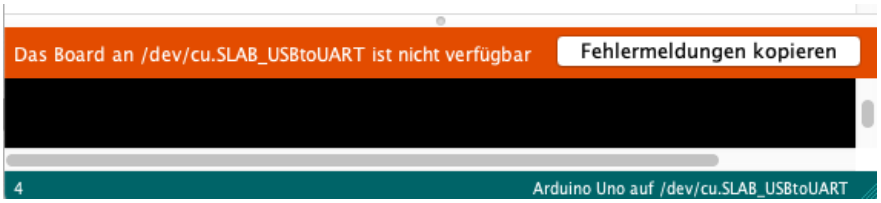
Ebenfalls im Menü **Werkzeuge** findest du den Eintrag **Port**. Wähle hier den USB-Port, an dem dein Arduino angeschlossen ist. Das ist zunächst alles, was du machen musst, um deinen Arduino mit der IDE programmieren zu können.

INFORMATIONEN UND FEHLERMELDUNGEN

Zuletzt schauen wir noch auf den unteren Bereich des Fensters. Hier findest du einen Bereich, in dem dir Informationen, Hinweise und Fehlermeldungen angezeigt werden. Wenn du zum Beispiel einen Sketch erfolgreich auf deinen Arduino geladen hast, siehst du etwas in dieser Art:



Eine Fehlermeldung sieht hingegen beispielsweise folgendermaßen aus.



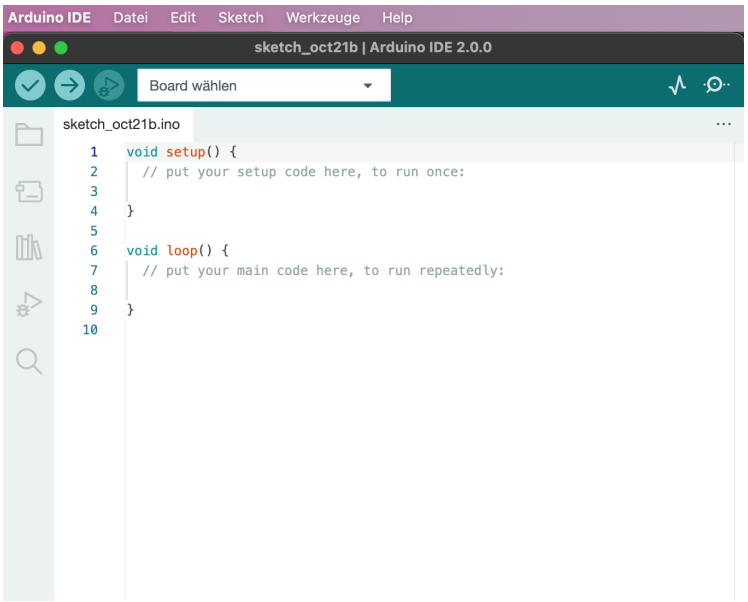
Hier ist das Problem ein nicht mit dem Rechner verbundener Microcontroller. Der Button **Fehlermeldung kopieren** ist oft praktisch. Damit kopierst du die Meldung in den Zwischenspeicher und kannst sie gleich darauf in einer Suchmaschine verwenden. Du wirst sehen, Suchmaschinen sind oft der beste Freund beim Programmieren – nicht nur als Anfänger.

In der nächsten Lektion wenden wir uns dem Code zu – und du wirst deinen ersten Sketch schreiben.

DIE ARDUINO IDE 2.0

Die neue IDE hat optisch das Rad nicht neu erfunden, es befinden sich lediglich einige Optionen, Tools und Einstellungen an anderen Orten. Ebenso macht die neue Version einen etwas frischeren Eindruck als die Oberfläche der Version 1.8.x

Wenn du die Arduino IDE 2.0 öffnest, erwartet dich dieses Fenster:



Auch hier findest du direkt die Buttons, um deinen Sketch zu überprüfen und auf den Arduino zu übertragen. Ebenso findest du rechts den Seriellen Monitor und daneben den Seriellen Plotter – ein Tool, das dir Daten, die dein Arduino sendet, grafisch anzeigen kann.

Allerdings kannst du oben über ein Dropdown-Menü direkt das angeschlossene Board auswählen und musst dafür nicht mehr das Menü öffnen. Linker Hand befindet sich eine Menüleiste, mit der du auf deine Dateien sowie die installierten Microcontroller und Bibliotheken. Über die Lupe kannst du im geöffneten Sketch suchen – eine Funktion, die du wie in vielen anderen Programmen auch mit Strg+F aufrufen kannst.

WAS IST NEU?

Zwei Dinge haben in die Version 2.0 Einzug gehalten, die es vorher in der Arduino IDE noch nicht gab: automatische Code-Vervollständigung und ein Debugger.

Ersteres musst du evtl. zunächst aktivieren. Öffne dazu die Einstellungen und setze neben **Schnelle Editor Vorschläge** einen Haken. Damit schlägt dir die IDE schon beim Tippen z.B. Variablen vor, die du bereits deklariert hast, wie auf diesem Screenshot zu sehen:

Das kann äußerst hilfreich sein und verhindern, dass du die gleiche Variable versehentlich unterschiedlich schreibst.

Mit dem Debugger (du findest ihn in der Menüspalte links – der Play-Button mit dem Käfer bzw. englisch “Bug”) kannst du deinen Code Zeile für Zeile ausführen und dich damit auf Fehlersuche begeben. Dieses Tool steht dir leider nicht für den Arduino UNO zur Verfügung, weswegen wir es in diesem

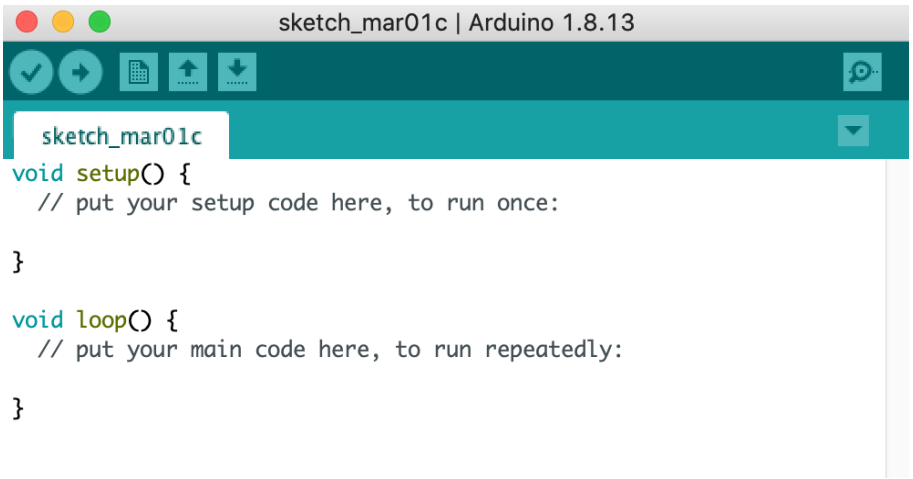
Buch nicht ausführlicher behandeln. Mehr Informationen zum Debugger und mit welchen Boards er funktioniert, findest du auf der offiziellen Arduino-Seite hierzu.

3 DEIN ERSTER SKETCH

Jetzt wird es Zeit für die Praxis! In den folgenden Lektionen lernst du das Grundgerüst eines Arduino-Sketchs kennen und schreibst deine eigenen Programme. Los geht's!

SETUP UND LOOP

Sobald du einen neuen Sketch in der Arduino IDE anlegst und sich das Fenster öffnet, siehst du immer folgendes Grundgerüst:



Aber was verbirgt sich dahinter? Wie du vielleicht schon weißt, verwendest du in der Arduino IDE die Programmiersprache C++. In dieser Sprache lassen sich – so wie übrigens in allen anderen auch – sogenannte **Funktionen** definieren.

Diese gehören eigentlich zum fortgeschrittenen Programmieren. Du kommst aber aufgrund der Struktur eines Arduino-Sketchs nicht drumherum, Funktionen zumindest in ihren Grundzügen zu verstehen.

Mit Funktionen kannst du deinen Code strukturieren, indem du ihnen voneinander getrennte Aufgaben zuweist. Hierbei **definierst** du zunächst die Funktion nach einem festen Muster. In C++ sieht dieses folgendermaßen aus:

```
Typ Name() {
  Code;
}
```

Der Typ gibt an, um welche “Sorte” von Funktion es sich handelt. In unserem Fall ist dies der Typ **void**. Das bedeutet, dass diese Funktion nichts weiter tut, als den Code zwischen den geschweiften Klammern **{ }** auszuführen.

Der Name ist – richtig, der Name der Funktion. In unserem Fall also **Setup** und **Loop**. Innerhalb der beiden geschweiften Klammern kommt dann alles, was du diese Funktionen ausführen lassen möchtest.

Hier gleich ein Hinweis: Achte immer darauf, dass jede öffnende Klammer **{** auch ein schließendes Gegenstück **}** hat. Ansonsten wird dein Sketch nicht ausgeführt. **Aber keine Angst, solltest du eine Klammer vergessen, weist dich die Arduino IDE vor dem Hochladen des Sketchs darauf hin.**

Aber wozu sind diese beiden Funktionen gut?

DIE SETUP-FUNKTION

In dieser Funktion bestimmst du, was dein Arduino nach dem Start **einmal** ausführen soll. Hier kannst du zum Beispiel festlegen,

- ob ein Pin Signale sendet oder empfängt.
- ob eine LED zu Beginn des Programms aus- oder eingeschaltet sein soll.
- dass der Serielle Monitor in einer bestimmten Geschwindigkeit laufen soll.

All das muss dein Arduino nur einmal zu Beginn des Sketchs wissen – es ist also, wie der Name der Funktion schon sagt, das **Setup**.

Später wirst du die Setup-Funktion mit Code füllen. Zunächst jedoch ein Blick auf den **Loop**.

DIE LOOP-FUNKTION

Im Gegensatz zum Setup wird der gesamte Code innerhalb des Loops ständig wiederholt. Dein Arduino führt ihn Zeile für Zeile aus, bis er am Ende angekommen ist – und beginnt dann wieder von vorne.

Das machst du dir gleich zunutze, indem du mit der Loop-Funktion die interne LED deines Arduino blinken lässt.

LASS DIE LED DES ARDUINOS BLINKEN

Zeit für dein erstes Programm! Auf deinem Arduino UNO befinden sich mehrere kleine LEDs – eine davon wirst du nun immer wieder ein- und ausschalten bzw. blinken lassen.

Öffne zunächst die Arduino IDE und erstelle einen leeren Sketch. Wähle hierfür im Menü Datei den Punkt Neu. Nun siehst du die beiden leeren Funktionen void setup() und void loop().

DIE SETUP-FUNKTION

Zunächst widmen wir uns der Setup-Funktion. Trage hier zwischen die beiden geschweiften Klammern **{ }** folgende Zeile Code ein:

```
pinMode(LED_BUILTIN, OUTPUT);
```

Die Setup-Funktion sieht dann so aus:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(LED_BUILTIN, OUTPUT);
}
```

Schauen wir uns diese Zeile genauer an. Zunächst gibt es hier eine weitere Funktion: **pinMode()**. Auch sie führt eine ganz bestimmte Aktion aus – sie legt für einen bestimmten Pin deines Arduinos eine Richtung bzw. ihren **Modus** fest: **Entweder ein Signal senden (OUTPUT) oder ein Signal empfangen (INPUT).**

Im Gegensatz zur Setup- und Loop-Funktion "erwartet" diese Funktion jedoch etwas zwischen den runden Klammern **( )**, nämlich sogenannte **Parameter.**

Der erste dieser Parameter bestimmt den Pin, dessen Richtung festgelegt werden soll. In unserem Fall ist das kein Analog- oder Digital-Pin an den Seiten des Arduinos, sondern die interne LED. Diese wird im Sketch mit **LED_BUILTIN** bezeichnet.

Der zweite Parameter bestimmt dann die Richtung – entweder OUTPUT oder INPUT. Wenn du eine LED steuern möchtest, muss der Arduino ihr entsprechende Signale senden. Vom Arduino aus gesehen ist das also ein OUTPUT. Wenn jedoch zum Beispiel ein Sensor Messdaten an den Arduino sendet, ist das wiederum ein eintreffendes Signal – also ein INPUT.

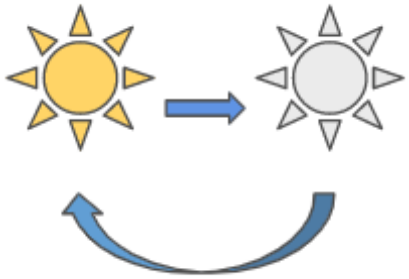
*Hinweis: Achte darauf, dass du jede Zeile, die nicht mit einer geschweiften Klammer endet, mit einem Semikolon **;** abschließt.*

Ansonsten kann dein Sketch nicht hochgeladen werden. Von dieser Regel gibt es Ausnahmen, die jedoch jetzt noch nicht wichtig sind.

Die Richtung des Pins musst du nur einmal zu Beginn deines Programms festlegen. Deshalb befindet sich die Funktion **pinMode()** in der Setup-Funktion.

DIE LOOP-FUNKTION

Kommen wir zum Loop. Du weißt, dass sich der Code innerhalb des Loops ständig wiederholt. Das können wir uns für das Blinken der LED zunutze machen, denn **eigentlich ist Blinken nichts anderes als 1) das Licht anschalten und 2) das Licht ausschalten – und wieder von vorne:**



DIE LED EINSCHALTEN

Trage zunächst in deiner Loop-Funktion zwischen den geschweiften Klammern **{ }** folgende Zeile ein:

```
digitalWrite(LED_BUILTIN, HIGH);
```

Hierbei handelt es sich wieder um eine Funktion, diesmal also **digitalWrite()**. Diese Funktion kann über einen Digital-Pin entweder das Signal **HIGH** oder **LOW** senden. HIGH bedeutet so viel wie “an” bzw. 1 – LOW hingegen steht für “aus” bzw. 0.

Am Beispiel einer LED ist das recht leicht zu verstehen: HIGH schaltet die LED ein, LOW schaltet sie aus.

Auch die Funktion **digitalWrite()** erwartet zwei Parameter: Den Pin, den sie ansteuern soll, und das Signal. In unserem Fall ist das also der Pin **LED_BUILTIN** und zunächst das Signal "Einschalten" – also **HIGH**.

Du hast nun also eine Zeile, um die LED einzuschalten. Aber das ist natürlich noch kein ordentliches Blinken.

WARTE KURZ

So ein Arduino ist erstaunlich schnell. Würdest du die LED einschalten und sie sofort danach wieder ausschalten, würdest du vom Blinken nichts mitbekommen. Der Wechsel von An und Aus wäre so schnell, dass es so aussähe, als würde die LED durchgängig leuchten.

Wir müssen also kurz warten.

Im Sketch funktioniert das mit der Funktion **delay()**. Diese Funktion verzögert sozusagen den weiteren Ablauf deines Programms. **Hierfür erwartet sie einen einzigen Parameter, nämlich die Dauer der Verzögerung – in Millisekunden**.

Trage als nächstes folgende Zeile in der Loop-Funktion ein:

```
delay(1000);
```

Damit verzögerst du die weitere Ausführung um 1.000 Millisekunden, was genau einer Sekunde entspricht. **Die Verzögerung sorgt dafür, dass die LED, die du in der Zeile davor angeschaltet hast, eine Sekunde lang brennt. Erst dann wird die nächste Zeile ausgeführt**.

DIE LED AUSSCHALTEN UND WIEDER WARTEN

In den nächsten zwei Zeilen Code drehst du den Spieß um – zunächst schaltest du die LED aus und sorgst dafür, dass das für eine Sekunde so bleibt:

```
digitalWrite(LED_BUILTIN, LOW);
delay(1000);
```

Die Funktion **digitalWrite()** kennst du ja bereits. Diesmal sendet sie an die interne LED (LED_BUILTIN) jedoch das Signal LOW. Sie schaltet sie also aus – und zwar ebenfalls für eine Sekunde.

Die vollständige Loop-Funktion sieht dann folgendermaßen aus:

```
void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

ALLES WIEDER VON VORNE

Jetzt hast du alles, was du für deine blinkende LED brauchst: Du schaltest das Licht kurz ein und schaltest es wieder kurz aus. **Den Rest, nämlich die Wiederholung, erledigt die Loop-Funktion von ganz alleine.**

Sobald dein Programm das zweite Mal **delay(1000);** abgearbeitet hat, ist es am Ende des Loops angekommen und springt wieder zu dessen Anfang – also in die Zeile **digitalWrite(LED_BUILTIN, HIGH);**

Probiere es gleich aus! Kopiere den folgenden Sketch in deine Arduino IDE und lade ihn auf deinen Arduino. Blinkt die LED? Spiele nun etwas mit den Parametern in der Delay-Funktion herum und verändere sie. Vielleicht kannst du zur Übung ja sogar etwas morsen, indem du die LED unterschiedlich lang aufleuchten lässt.

```
void setup() {
  // put your setup code here, to run once:
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

FINDET DEINE IDE DEN ARDUINO UNO NICHT?

Wenn das der Fall ist, fehlt in deiner Arduino IDE noch der passende Treiber für den Chip auf deinem Arduino UNO. Aber das ist kein Problem, du kannst den Treiber ganz einfach nachinstallieren:

FÜR WINDOWS

Lade dir zunächst hier den Treiber herunter und entpacke die .ZIP-Datei. Führe nun die Installationsdatei aus – nach einem Neustart der IDE findest du deinen (angeschlossenen) Arduino UNO unter Werkzeuge > Boards.

FÜR MAC

Den Treiber findest du hier auf Github. Öffne die .PKG-Datei und folge den Anweisungen des Installations-Wizards. Nach der Installation musst du deinen Mac neu starten. Prüfe nun zunächst, ob du deinen (angeschlossenen) Arduino UNO in der IDE unter Werzeuge > Boards findest. Falls nicht, musst du den neuen Treiber noch aktivieren. Hierfür findest du weitere Instruktionen auf der Github-Seite des Entwicklers unter Installation.

FÜR LINUX

Wenn der passende Treiber dir noch nicht zur Verfügung steht, musst du nur ein Update deines Betriebssystems machen. 🙂

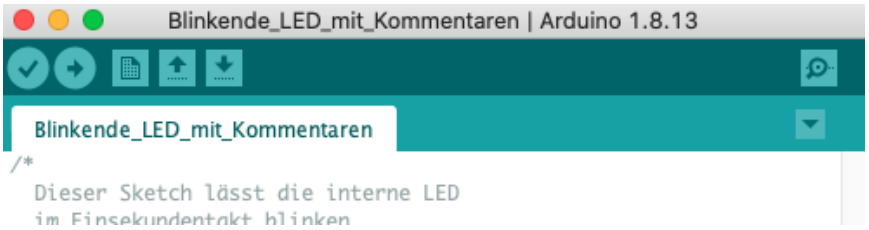
Weitere Updates und Informationen zum Treiber findest du hier.

4 DER SERIELLE MONITOR

Kommen wir nun zum Seriellen Monitor – einem Feature der Arduino IDE, das du so gut wie bei jedem deiner Projekte verwenden wirst.

ÜBERBLICK

Erinnerst du dich an die Lupe oben rechts in deinem Sketch-Fenster? Mit einem Klick hierauf öffnest du den Seriellen Monitor – **sofern dein Arduino UNO am Rechner angeschlossen ist und du den richtigen USB-Port ausgewählt hast.**



Der Serielle Monitor ist so etwas wie die Kommunikationsschnittstelle deines Arduino. Er kann hier Messwerte und andere Daten ausgeben, sodass du sie hier überprüfen kannst. Aber diese Schnittstelle funktioniert

auch in die andere Richtung. Auch du kannst deinem Arduino über den Seriellen Monitor Befehle senden.

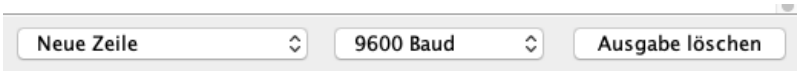
Werfen wir zunächst einen Blick auf die wichtigsten Teile des Monitors.

Ganz oben in der Titelleiste findest du den USB-Port, über den der Serielle Monitor mit dem Arduino kommuniziert.

Darunter befindet sich ein Eingabefeld samt Button zum **Senden**. Hierüber kannst du deinem Arduino Daten und Befehle senden. Zunächst bleiben wir jedoch beim Lesen.

Der größte Teil des Fensters ist für die Ausgabe reserviert – hier wirst du in der nächsten Lektion einen Text anzeigen lassen.

Unten links findest du zwei Checkboxen: Einmal **Autoscroll** – damit bewegt sich der Serielle Monitor mit den Ausgabedaten mit. Und **Zeitstempel anzeigen** – damit werden alle Daten, die du mit dem Arduino an den Monitor sendest, mit der Uhrzeit versehen, an der sie dort eintrafen.



Unten rechts findest du ein Dropdown, in dem standardmäßig **Neue Zeile** steht. Dieses bezieht sich wieder auf das Senden von Daten an deinen Arduino. **Daneben befindet sich jedoch noch eine wichtige Einstellung, die oft eine Fehlerquelle ist:**

DIE BAUDRATE

Die Baudrate (oder auch Symbolrate) bezeichnet die Menge an Zeichen, die pro Sekunde übertragen werden. **Oft wird in Projekten mit einem Arduino UNO eine Rate von 9.600 verwendet.**

Wenn du einmal das Dropdown-Menü öffnest, siehst du, dass es hier noch viele weitere Baudraten zur Auswahl gibt. Wenn du später einmal Projekte mit dem ESP8266 baust, werden diese für dich interessant, da dieser Microcontroller deutlich schneller als ein Arduino UNO ist.

Eine Sache ist wichtig: In diesem Dropdown-Menü muss immer die gleiche Baudrate wie in deinem Sketch eingestellt sein (Wie du sie dort einstellst, erfährst du demnächst). Wenn sich die Baudraten im Sketch und im Monitor unterscheiden, kann **nichts kaputtgehen** – aber du siehst oft einen ziemlichen Zeichensalat.

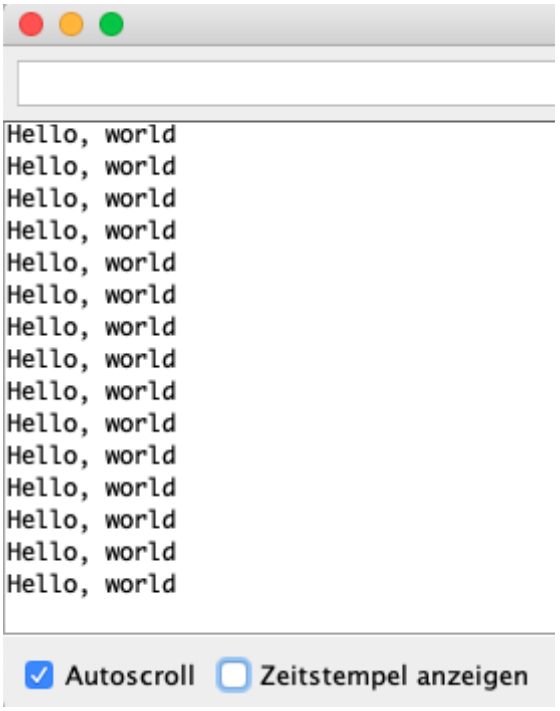
Bleibt noch ein einziger Button übrig: **Ausgabe löschen**. Wie der Name schon sagt, löschst du damit den Inhalt des Ausgabefensters

Noch ein Hinweis: Dein Arduino kann nicht nur über USB kommunizieren, sondern auch über die Pins 0->RX und 1<-TX. Deshalb solltest du an diese Pins nichts anschließen, außer du möchtest sie später einmal für die serielle Kommunikation verwenden.

Jetzt steigst du in die Praxis ein, aktivierst den Seriellen Monitor und gibst deine ersten Zeichen dort aus.

5 HELLO, WORLD

Zeit für den Klassiker der Programmierung – **hello, world**. Du schreibst nun ein kleines Programm, das in deinem Seriellen Monitor immer wieder die Zeile “hello, world” ausgibt.



DIE VERBINDUNG ZUM SERIELLEN MONITOR

Wie du den Seriellen Monitor im Sketch-Fenster öffnest, weißt du bereits (mit der Lupe oben rechts). Bevor dort jedoch Daten erscheinen können, musst du ihn auch in deinem Sketch starten – eine einmalige Angelegenheit, weshalb der Code hierfür in die Setup-Funktion kommt:

```
void setup() {
  Serial.begin(9600);
}
```

Die Funktion **Serial** erweiterst du hier um den Befehl begin(), um die Verbindung einzurichten. **Achte unbedingt auf den Punkt zwischen Serial und begin.**

Als Parameter zwischen den runden Klammern gibst du der Funktion die Baudrate mit – in unserem Fall also 9600. Wie du bereits weißt, muss diese mit der Einstellung im Seriellen Monitor übereinstimmen.

ZEICHEN IM SERIELLEN MONITOR AUSGEBEN

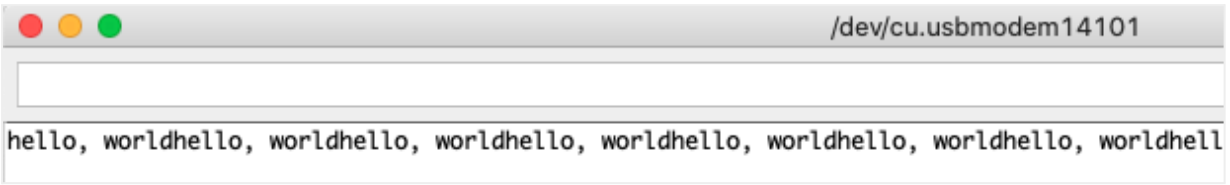
Nun kommt der interessante Teil. Trage im Loop folgenden Code ein:

```
Serial.print("hello, world");
Serial.println();
delay(1000);
```

In der ersten Zeile siehst du den Befehl **.print()**. Zwischen den Klammern steht die Zeichenkette (String) **hello, world**. Wie du siehst, steht dieser kleine Gruß zwischen Anführungsstrichen – **das ist besonders wichtig und eine häufige Fehlerquelle.**

Wenn du die Anführungsstriche testweise entfernst, wirst du eine Fehlermeldung erhalten. Das liegt daran, dass dann **hello** als Variable interpretiert wird – und nicht als String. Dem Thema Variablen wenden wir uns in einer späteren Lektion zu.

In der Zeile darunter steht **Serial.println();** – hiermit erzeugst du im Seriellen Monitor eine neue Zeile. Würdest du das nicht tun, würde dein Arduino die Zeichen einfach nur hintereinander schreiben:



Du kannst diese beiden Befehle – also den Gruß und die neue Zeile – jedoch auch kombinieren und dir so eine Zeile Code sparen:

```
Serial.println("hello, world");
```

Zuletzt findest du im Loop noch einen Delay von einer Sekunde. Hast du den Sketch schon ausprobiert? Erscheint der Gruß in deinem Seriellen Monitor?

In der nächsten Lektion behandeln wir eines der wichtigsten Themen: Variablen.

6 VARIABLEN

Um Variablen kommt kein Programmierer herum. In diesem Abschnitt lernst du, welche Typen es gibt und was du damit anstellen kannst.

WOZU SIND SIE GUT?

Variablen sind – wie ihr Name schon sagt – variabel, also veränderlich. Genauer gesagt ist es ihr Inhalt, der veränderlich ist.

Du kannst zum Beispiel die Zahl 10 in deinem Sketch hinterlegen, um sie im Seriellen Monitor auszugeben – so wie du es mit dem Text **hello, world** gemacht hat.

```
Serial.print(10);
```

Du kannst aber auch eine Variable verwenden und die Zahl dort speichern:

```
int zahl = 10;
```

Wenn du den Inhalt dieser Variablen dann in einem Seriellen Monitor ausgeben möchtest, ersetzt du diese “feste” Zahl durch die Variable:

```
Serial.print(zahl);
```

Im Seriellen Monitor siehst du keinen Unterschied – das Ergebnis ist dasselbe. **Beim Programmieren macht das jedoch einen großen Unterschied!** Stell dir vor, du möchtest die Zahl an mehreren Stellen in deinem Sketch verwenden. Eines Tages möchtest du sie jedoch durch eine 11 ersetzen. Das würde bedeuten, dass du jede einzelne Stelle im Sketch, an der sie eingetragen ist, ändern müsstest. Wenn du jedoch eine Variable verwendest, musst du das nur einmal tun.

```
int zahl = 11;
```

Überall wo du dann die Variable eingesetzt hast, verwendet sie deine neue Zahl.

In C++ gibt es eine Vielzahl von Variablentypen, zum Beispiel für Texte (Strings), ganze Zahlen und Kommazahlen. In der nächsten Lektion schauen wir uns diese Typen genauer an.

WELCHE TYPEN GIBT ES?

In C++ gibt es mehrere Variablentypen, die du ganz unterschiedlich einsetzen kannst. Bleiben wir zunächst beim Text **hello, world**. Dieser Text soll nun in einer Variablen für Texte gespeichert werden.

Am einfachsten funktioniert das in der Arduino IDE mit dem Typ **String**. Und damit machen wir einen kurzen Abstecher zur Deklaration von Variablen.

STRINGS DEKLARIEREN

Die Deklaration von Variablen funktioniert recht einfach und immer nach dem gleichen Prinzip:

```
Typ Name = Wert;
```

In unserem Fall wäre das also

```
String text = "hello, world";
```

Lass uns kurz über den Namen von Variablen sprechen. Im Prinzip sind hier deiner Phantasie keine Grenzen gesetzt. Es gibt jedoch ein paar Namen, die du nicht verwenden kannst, da sie schon von anderen Funktionen in C++ besetzt sind.

Dazu gehören zum Beispiel: **if, else, continue, class** und **true.**

Die Liste der verbotenen Wörter ist lang – **du musst sie dir jedoch zum Glück nicht merken.** Immer wenn du einen Variablennamen bei der Deklaration verwendest, der schon anderweitig besetzt ist, färbt sich der Name ein (erlaubte Namen bleiben schwarz) und spätestens beim Hochladen des Sketchs erhältst du eine Fehlermeldung.

Außerdem darfst du Variablennamen nicht mit einer Zahl oder einem Sonderzeichen beginnen.

Dafür kannst du die Namen jedoch groß oder klein schreiben. Es ist jedoch üblich, einen Variablennamen klein zu beginnen. Wenn der Name aus zwei oder mehr Wörtern besteht, kommt der sogenannte **Camel Case** zum Einsatz. Hier beginnst du das erste Wort klein und schließt alle weiteren Wörter mit einem Großbuchstaben an:

```
meineKamelVariable
```

Mit viel Phantasie erkennst du hier die Höcker eines Kamels, naja… **Noch ein letzter Tipp:** Versuche, Variablen Namen zu geben, die ihren Zweck gut beschreiben. Das hilft dir und anderen Lesern deines Codes zu verstehen, was darin gespeichert wird.

WEITERE TYPEN VON VARIABLEN

Neben Texten gibt es natürlich auch Zahlen. Hierfür gibt es gleich mehrere Typen, je nachdem um welche Zahl es sich handelt:

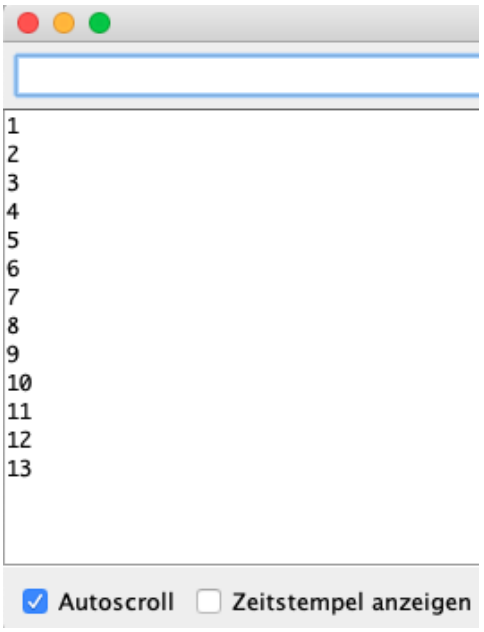
Typ	Geeignet für	Zahlenbereich
int	Ganze Zahlen	-32.768 bis 32.767
long	Ganze Zahlen	-2.000.000.000 bis 2.000.000.000
float	Fließkommazahlen	-3.4028235E+38 bis 3.4028235E+38 (ja, sehr groß)

Im weiteren Verlauf des Kurses wirst du noch Variablen des Typs **int** – also ganze Zahlen – verwenden. Und noch ein ganz besonderer Typ: **bool**. Variablen des Typs **bool** können nur zwei Werte, bzw. Zustände annehmen: True (wahr) oder False (falsch).

In der nächsten Lektion geht es weiter mit den Variablen – und zwar in der Praxis.

VARIABLEN IN AKTION

Als nächstes baust du dir einen Zähler im Seriellen Monitor, der beginnend bei Eins jede Sekunde eins hochzählt.



Schauen wir uns zunächst an, wie du eine Variable für ganze Zahlen deklarierst. Hierfür benötigst du den Variablentyp **int** – was übrigens für **integer**, also Ganzzahl steht. Angenommen die Variable soll **zahl** heißen und zunächst den Wert **1** besitzen:

```
int zahl = 1;
```

Das war einfach. Du fragst dich jedoch vielleicht, an welcher Stelle im Sketch diese Zeile stehen soll – und hier kommt ein weiterer wichtiger Aspekt von Variablen ins Spiel: **Scope**.

GLOBALE UND LOKALE VARIABLEN

Prinzipiell gibt es zwei Zustände, die Variablen hinsichtlich ihres Scopes einnehmen können: **global** und **lokal**. Global bedeutet, einfach gesagt, dass die Variable für alle Funktionen im Sketch zur Verfügung steht und auch dort verändert werden kann. Lokale Variablen können hingegen nur in der Funktion verwendet werden, in der sie deklariert wurden – und auch nur dort gelesen und verändert werden.

Zwei dieser Funktionen kennst du ja bereits aus jedem Sketch, den du neu anlegst: **void setup()** und **void loop()**. Würdest du die Variable nun in der Setup-Funktion anlegen, wäre sie auch nur dort verfügbar, da sie **lokal** wäre:

```
void setup() {
  int zahl = 1;
  Serial.begin(9600);
}

void loop() {
  Serial.println(zahl);
}
```

Das bedeutet konkret, dass der Loop die Variable **zahl** nicht “sehen” – und sie deshalb auch nicht im Seriellen Monitor ausgeben könnte. Das bedeutet also, dass wir eine **globale** Variable brauchen. **Diese deklarierst du ganz zu Beginn des Sketchs, also noch vor dem Setup:**

```
int zahl = 1;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(zahl);
}
```

So stellst du sicher, dass der Loop auf sie zugreifen kann. Kehren wir zurück zum Zähler. Hierfür möchten wir zunächst die Variable **zahl** im Seriellen Monitor ausgeben. Anschließend soll die Zahl in der Variablen um eins erhöht werden. Das funktioniert ganz einfach:

```
zahl = zahl + 1;
```

Irritiert dich diese Schreibweise? Dann versuche die Zeile Code einmal so zu lesen: *"Nimm die Variable **zahl**, weise ihr den bisherigen Wert in der Variable zu und addiere hierzu die Zahl 1."*

Es gibt noch eine weitere, kürzere Schreibweise – von der die Sprache C++ übrigens auch ihren Namen hat:

```
zahl++;
```

Diese Schreibweise ist auf den ersten Blick zwar weniger intuitiv, spart dir aber beim Programmieren etwas Zeit.

Übrigens, wenn du statt der Addition lieber andere Rechenarten verwenden möchtest, funktioniert das so:

```
zahl = zahl -1; //Subtraktion
zahl--; //Auch Subtraktion
zahl = zahl * 2; //Multiplikation
zahl = zahl / 2; //Division
```

Und das ist alles, was du für den Zähler benötigst. Der vollständige Sketch sieht dann folgendermaßen aus:

```
int zahl = 1;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(zahl);
  zahl++;
  delay(1000);
}
```

Auch hier machst du dir wieder den Loop zunutze: Er gibt die Variable **zahl** aus, addiert eins hinzu, wartet eine Sekunde – und beginnt diese Prozedur wieder von vorne. So lange, bis du den Arduino neu startest oder ausschaltest.

In den nächsten Lektionen wenden wir uns der Hardware zu – beginnend mit einer LED.

7 EINE LED STEUERN

Jetzt wird es Zeit für etwas mehr Hardware! In den nächsten Lektionen schließt du eine LED an deinem Arduino an. Diese schaltest du dann in deinem Sketch an und aus. Danach baust du dir einen Dimmer, mit dem du die Helligkeit der LED regelst.

SO SCHLIESST DU EINE LED AN

Bevor es losgehen kann, musst du zunächst deine LED mit dem Arduino UNO verbinden. Hierfür benötigst du neben der LED einen Widerstand mit 220Ω, ein Breadboard und drei Kabel.

Zunächst ein paar Worte zum Breadboard (Steckplatine). Hierauf kannst du Bauteile platzieren, indem du sie einfach hineinsteckst. Unter den Löchern

verlaufen leitfähige Leisten, die den Strom zwischen den Löcher fließen lassen:

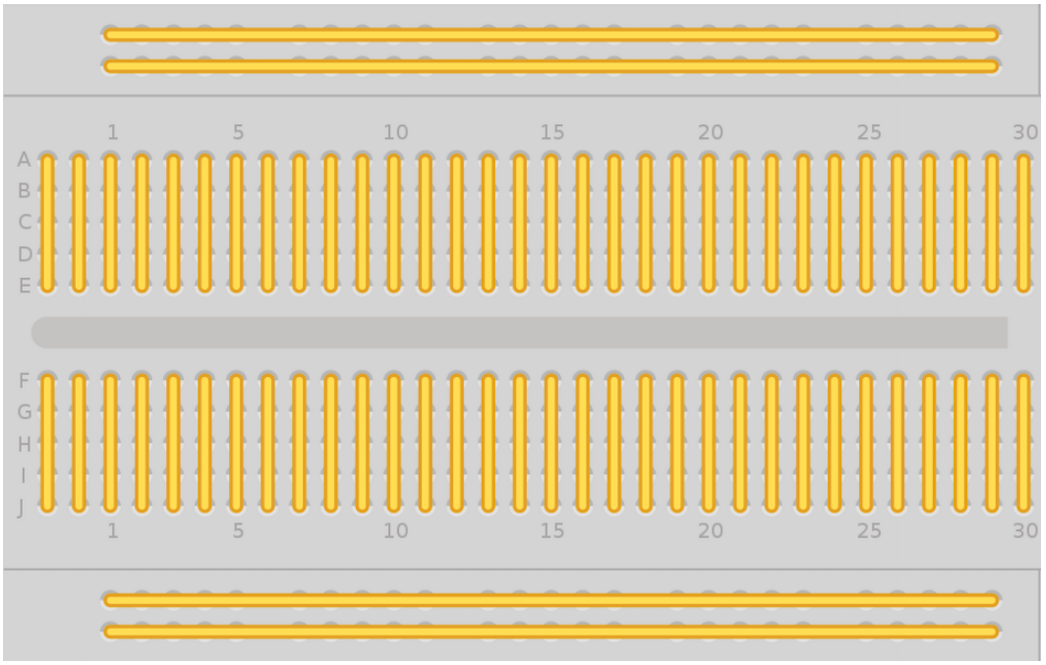
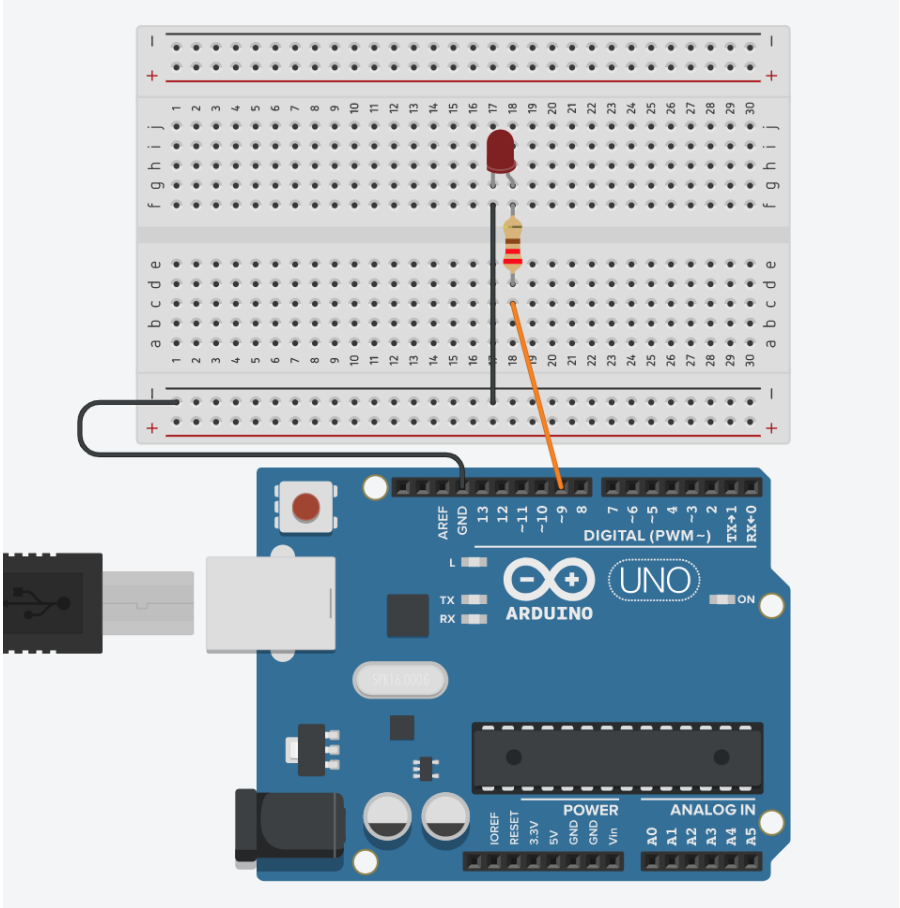


Bild: Andreas B Mundt – Eigenes Werk, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=19620117

Wie du siehst, verläuft der Strom oben und unten auf zwei Leisten quer über das Breadboard. Diese sind auf den meisten Boards mit **+** und **–** gekennzeichnet. Diese Leisten verbindest du zum Beispiel mit den Pins **5V** und **GND** deines Arduinos, sodass du dann von den Leisten aus Strom für deine Bauteile ableiten kannst.

Unter den Löchern im Zentrum sitzen kleinere Leisten, die quer dazu angeordnet sind. Hier läuft der Strom immer unterhalb der fünf Löcher in einer Reihe. In der Mitte des Breadboards befindet sich eine “Brücke”, die es in zwei voneinander getrennte Bereiche aufteilt, über die kein Strom hinweg fließt.

Zurück zur LED: Orientiere dich beim Aufbau an folgender Skizze:



Screenshot: Tinkercad

Sprechen wir zunächst über die LED. Diese hat zwei verschieden lange Beine – das längere heißt **Anode** und ist der Pluspol. Das kürzere ist die **Kathode**, der Minuspol. Damit die LED leuchtet, muss Strom durch sie fließen.

Allerdings benötigst du für jede LED einen sogenannten **Vorwiderstand**. Das liegt daran, dass eine LED immer so viel Strom von der Quelle zieht, wie sie bekommen kann – **sie kann ihren Verbrauch nicht selbst regulieren.** Je länger sie brennt, desto leitfähiger wird sie. Das bedeutet, dass auch ihr Stromverbrauch immer weiter ansteigt. Hier kommt der Widerstand ins Spiel, der selbst eine gewisse Menge Strom verbraucht und der LED nur die Menge Strom übrig lässt, mit der sie zurechtkommt.

Würdest du keinen Widerstand zwischen Arduino und LED einbauen, würde sie mit der Zeit immer mehr Strom ziehen und damit heißer werden – bis sie schließlich durchbrennt.

Die Anode verbindest du – samt dazwischen geschaltetem Widerstand – mit dem Digital-Pin 9 deines Arduinos. Die Kathode verbindest du hingegen mit der Erde – also mit **GND**.

Und das ist auch schon alles, was du tun musst. Weiter geht es mit der Steuerung der LED.

DIE LED MIT EINEM SKETCH STEUERN

Jetzt wo deine LED am Arduino angeschlossen ist, möchtest du sie sicher auch einschalten. Kein Problem!

Für einen ersten Test lässt du die LED wieder blinken. Diesmal lässt du sie jedoch anfangs ganz schnell blinken und sie dann immer langsamer werden.

VARIABLEN UND DIE SETUP-FUNKTION

Zunächst deklarierst du zwei Variablen zu Beginn deines Sketchs – damit sie global, also überall im Sketch, verfügbar sind:

```
int ledPin = 9;
int pause = 0;
```

Die erste Variable **ledPin** legt den Pin fest, an dem die LED angeschlossen ist – in unserem Fall die 9. Die zweite Variable **pause** kommt im Loop ins Spiel und wird die Zeitspanne bestimmen, die die LED nicht leuchtet. Da diese immer größer werden soll, benötigst du hierfür eine Variable, in der du immer größere Zahlen abspeichern kannst.

In der Funktion **void setup()** musst du nichts weiter tun, als den **pinMode** des **ledPin** festzulegen. Da du Signale vom Arduino aus senden möchtest, ist das also **OUTPUT.**

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

DER LOOP

Hier verwendest du zunächst die Funktion **digitalWrite()**, die du bereits von der letzten blinkenden LED (der internen) kennst.

Du sendest ein **HIGH** an die LED, wartest dann 500 Millisekunden und schaltest sie mit einem **LOW** wieder aus:

```
digitalWrite(ledPin, HIGH);
delay(500);
digitalWrite(ledPin, LOW);
```

Jetzt wird es interessant! In der nächsten Zeile erhöhst du den Wert in der Variablen **pause** um 50. Beim ersten Durchlauf des Loops also von 0 auf 50 – im zweiten von 50 auf 100 – im dritten von 100 auf 150 – und immer so weiter.

Diesen Wert verwendest du dann in der Funktion **delay()** als Millisekunden, die die LED nicht brennen soll:

```
pause = pause + 50;
delay(pause);
```

Übrigens: Auch um eine Variable um einen anderen Wert als Eins zu erhöhen, gibt es eine Kurzform. Du kannst das auch so programmieren:

```
pause+= 50;
```

Lade den folgenden Sketch auf deinen Arduino und schaue, was deine LED macht. Wenn alles richtig angeschlossen ist, sollte sie anfangs sehr schnell blinken. Mit der Zeit werden die Abstände immer größer und das Blinken damit langsamer.

```
int ledPin = 9;
int pause = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  pause+= 50;
}
```

8 BEDINGTE ANWEISUNGEN & VERGLEICHE

Kommen wir zu einem weiteren wichtigen Konzept beim Programmieren: Bedingte Anweisungen – auch bekannt als **if…else…** Mit Hilfe dieser Anweisungen kannst du in deinem Sketch Abzweigungen für bestimmte Ereignisse einbauen.

Zum Einsatz kommt wieder die LED, die sich bereits auf deinem Breadboard befindet. Außerdem verwendest du wieder den Zähler, der jede Sekunde eine Zahl um 1 erhöht und diese im Seriellen Monitor ausgibt.

Die LED soll diesmal immer nur aufleuchten, wenn die aktuelle Zahl durch 3 teilbar ist. Bei allen anderen Zahlen bleibt sie aus.

Hierfür eignen sich bedingte Anweisungen. Aber eins nach dem anderen. Zunächst deklarierst du zu Beginn des Sketchs einige Variablen:

```
int ledPin = 9;
int zahl = 1;
int rest = 1;
```

Die ersten beiden kennst du schon. Die dritte Variable **rest** benötigst du, um zu testen, ob die aktuelle **zahl** durch 3 teilbar ist. Den Wert von **rest** setzt du zu Beginn auf 1.