Frederik Kumbartzki

VOM ANFÄNGER ZUM MAKER

Mit dem Arduino UNO & ESP32

Vorwort

Vielen Dank, dass du dich für dieses Buch entschieden hast.

Im Herbst 2022 erschien die erste Auflage von **Vom Anfänger zum Maker** und hat seitdem viele Tüftler begleitet. Nun ist es Spätsommer 2025 – es sind also gerade einmal knapp drei Jahre vergangen. Und doch hat sich in dieser Zeit viel getan, was in der Welt von Soft- und Hardware natürlich völlig normal ist.

So ist mittlerweile der Arduino UNO R4 erschienen, die Weiterentwicklung seines Vorgängers R3, der der ersten Auflage noch zugrunde lag. Zwar kannst du das aktuelle Board fast genauso verwenden wie den R3 – trotzdem gibt es einige Weiterentwicklungen, auf die ich in dieser Auflage eingehe.

Neben dem UNO R4 wurde auch die Software, mit der du deinen Arduino programmierst, aktualisiert. In der ersten Auflage war die Arduino IDE in der Version 2.0 noch eine Randnotiz – in diesem Buch steht sie im Mittelpunkt, denn sie ist mittlerweile zum Standard geworden.

Außerdem hat der ESP32 in dieser zweiten Auflage den ESP8266 abgelöst. Er ist leistungsstärker und bietet von Haus aus sowohl WLAN als auch Bluetooth, was ihn vielseitiger für IoT-Projekte macht. Daneben ist er in vielen Variationen erhältlich und in der Maker-Welt mittlerweile beliebter als der ESP8266.

Du findest in diesem Buch alle Sketches (Programme) für die vorgestellten Projekte. Da diese auf deinem Weg zum Maker immer länger und platzraubender werden, habe ich sie teilweise auf meine Webseite

ausgelagert, wo du sie kopieren und anschließend weiterverwenden kannst: **polluxlabs.net/maker-buch-2**

Ein paar Hinweise noch vorab: Die Bibliotheken, die du in deinen Programmen verwendest, sowie die Arduino IDE werden ständig aktualisiert. Das bedeutet leider auch, dass die vorgestellten Programme möglicherweise irgendwann nicht mehr wie abgedruckt funktionieren. Aber das ist in der Regel kein Problem, mit ein paar Anpassungen lassen sich diese Fehler meist schnell beheben. Updates zu diesem Buch und Fehlerbehebungen findest du ebenfalls auf der oben genannten Webseite.

Die Arbeit mit Strom ist grundsätzlich nicht ungefährlich. Bitte verwende zu deiner eigenen Sicherheit nur Strom aus Batterien oder von einem USB-Port. Batterien sind fachgerecht zu entsorgen.

Kinder und Jugendliche sollen an den hier vorgestellten Projekten und Tutorials nur unter Aufsicht eines Erwachsenen arbeiten.

Solltest du an einer Stelle dieses Buchs oder bei einem Projekt nicht weiterkommen, schreibe mir gerne eine E-Mail an info@polluxlabs.net

Diese Bauteile wirst du verwenden

Um die Projekte in diesem Buch nachbauen zu können, benötigst du einige elektronische Bauteile. Viele davon findest du in kostengünstigen Arduino Starter Kits. Andere, wie den Arduino UNO R4 oder den ESP32, musst du eventuell separat erwerben. Die folgende Liste gibt dir einen vollständigen Überblick.

I. Die Herzstücke: Microcontroller

Das sind die Gehirne deiner Projekte.

- Arduino UNO R4 (WiFi empfohlen): Die Basis für die meisten Projekte in diesem Buch. Die WiFi-Version wird für die Internet- und LED-Matrix-Projekte benötigt.
- Arduino Nano ESP32: Ein kompakter und leistungsstarker Controller mit WLAN und Bluetooth, der für die fortgeschrittenen IoT-Projekte zum Einsatz kommt.

II. Grundausstattung: Werkzeuge & Zubehör

Diese Teile brauchst du für fast jedes Projekt.

- Breadboard (Steckplatine): Die Grundlage für den Aufbau deiner Schaltungen.
- Jumper-Kabel (Steckbrückenkabel): In verschiedenen Längen und Ausführungen (Male-to-Male, Male-to-Female).
- USB-C-Kabel: Zum Anschließen und Programmieren des Arduino UNO R4 und des Nano ESP32.
- 9V-Blockbatterie mit Anschlussclip: Für die externe Stromversorgung von Motoren.

III. Grundlegende Bauteile: Widerstände & LEDs
Die absoluten Basics der Elektronik.

- Widerstände:
  - 220Ω: Standard-Vorwiderstände für LEDs.
  - 10kΩ: Sogenannte Pull-up- oder Pull-down-Widerstände für Buttons und Schalter.
- LEDs (Leuchtdioden): In verschiedenen Farben.
- RGB-LED: Eine LED, die Farben mischen kann.
- Kondensator (470µF oder mehr): Zur Stabilisierung der Stromversorgung für den Servo-Motor.

IV. Ein- & Ausgabegeräte: Sensoren, Displays & mehr
Hiermit interagieren deine Projekte mit der Welt.

- Potentiometer (Poti): Ein einfacher Drehregler.
- Piezo-Summer (Buzzer):
  - Passiver Piezo: Zum Erzeugen von Melodien.
  - Aktiver Piezo: Für einen einfachen Alarmton.
- Buttons: Einfache Druckknöpfe.
- Ultraschallsensor HC-SR04: Misst Entfernungen.
- Geräuschsensor (z.B. KY-037): Misst die Lautstärke der Umgebung.
- Temperatur- & Luftfeuchtigkeitssensor DHT11: Ein vielseitiger Sensor für Umgebungsdaten.
- LC-Display (16x2): Ein einfaches Textdisplay.
- 7-Segment Display (eine Ziffer): Eine Retro-Ziffernanzeige.
- Tilt-Switch (Neigungsschalter): Erkennt Bewegungen.
- Folientastatur (4x4): Für die Eingabe von Codes.
- OLED-Display (0.96 Zoll, SSD1306, I²C): Ein kleines, scharfes Grafikdisplay.

V. Motoren & ICs (Integrierte Schaltungen)
Jetzt kommt Bewegung ins Spiel!

- Gleichstrom-Motor (DC-Motor): Ein einfacher Motor für Rotationen.
- Servo-Motor (z.B. SG90): Ein Motor, der präzise Positionen ansteuern kann.
- H-Brücke L293D: Ein Chip zur Ansteuerung von Gleichstrom-Motoren.

Mit diesen Bauteilen kannst du sämtliche Projekte dieses Buchs durcharbeiten – und bist auch danach für viele weitere deiner Ideen bestens gerüstet. Neben der Hardware benötigst du noch einen Computer (PC oder Mac), auf dem du die Arduino IDE installieren kannst und Zugang zu einem WLAN-Netz.

Hast du alles parat? Dann lass uns loslegen!

1 Der Arduino UNO

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

Auch heute ist es für viele angehende Maker immer noch der erste Microcontroller, den sie in Händen halten. Er ist erschwinglich und die Arbeit mit ihm ist komfortabel. Das galt schon für die Version R3 und weiterhin für die aktuelle Version R4.

UNO R3 und UNO R4

Viele Jahre war der UNO R3 das Aushängeschild aller Arduinos. Er war so beliebt, dass es neben dem Original zahlreiche sogenannte Klone gab und immer noch gibt, die deutlich günstiger sind, bei denen man jedoch auch ein paar kleinere Abstriche bei der Handhabung machen muss: Die Buchsen der Pins sind oft etwas hakeliger und einige Boards benötigen einen speziellen Treiber (CH340) für die Kommunikation mit deinem Computer, den du vorab manuell installieren musst.

Ich persönlich empfehle dir, Original-Hardware von Arduino zu verwenden. Hier kannst du auf eine sorgfältige Verarbeitung und sehr gute Kompatibilität vertrauen, was sicherlich ein paar Euro mehr wert ist.

Im Jahr 2023 gab es für den UNO jedoch ein lange erwartetes Update, das das Board auf ein neues Level hob. Neben einem neuen, leistungsstärkeren Prozessor verfügt der UNO R4 nun (auf Wunsch) auch über WLAN und Bluetooth. Damit eignet er sich nun auch für Projekte im Bereich **Internet of Things**, was vorher nicht so einfach möglich war.

Während der R3 einen 8-Bit ATmega328 nutzt, ist der R4 mit dem leistungsfähigeren Microcontroller RA4M1 Arm® Cortex®-M4 ausgestattet, der mit 48 MHz statt 16 MHz läuft. Der R4 bietet außerdem 32 KB RAM, was sechzehnmal mehr ist als beim R3, und 256 KB Flash-Speicher, was achtmal mehr Kapazität bedeutet.

Das Board gibt es in zwei Varianten: **R4 WiFi** und **R4 Minima**. Letzterer verfügt weder über WLAN noch Bluetooth – und hat auch nicht die schicke LED-Matrix, die du in deinen Projekten als Mini-Display verwenden kannst. Deshalb kostet der Minima auch ein paar Euro weniger als sein besser ausgestattetes Pendant. Ich denke jedoch, dass sich dieser kleine Aufpreis durchaus lohnt, und würde dir deshalb empfehlen, im Zweifelsfall zum R4 WiFi zu greifen.

Die neuen Möglichkeiten, die uns der UNO R4 WiFi mit seiner WLAN-Funktion und der LED-Matrix bietet, werden wir später in diesem Buch erkunden.

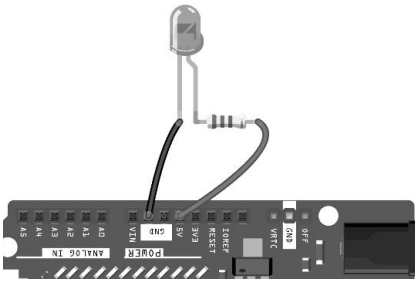
Die wichtigsten Pins des Arduino UNO

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

Power- und Analog-Pins

Zunächst die 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.

Auf dem folgenden Bild 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 Bauteile – wie die LED oben – 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 (Aus) oder eine Eins (An). 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 (RX/TX) sind für die serielle Kommunikation reserviert und können nicht programmiert werden.

Digital-Pins

Mit den Digital-Pins kannst du die Signale Null und Eins messen und ausgeben. Das bedeutet zum Beispiel, dass du eine LED über einen der Pins an- und ausschalten kannst. Oder ermitteln kannst, ob ein Button 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 erwartet hättest: **Du kannst hierüber ein sich veränderndes Signal ausgeben.** Damit kannst du zum Beispiel eine LED mit unterschiedlicher Helligkeit leuchten lassen. Der Fachbegriff hierfür heißt Pulsweitenmodulation (PWM).

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.

Eine weitere Möglichkeit ist der Pin VIN, den wir oben schon kurz angeschaut haben. Auch hierüber kannst du Batterien anschließen. Den Pluspol verbindest du hierbei 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 und Informationen ausgeben und z.B. 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.

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 der Arduino IDE unter **www.arduino.cc/en/software** findest du die jeweils aktuelle Version.

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.

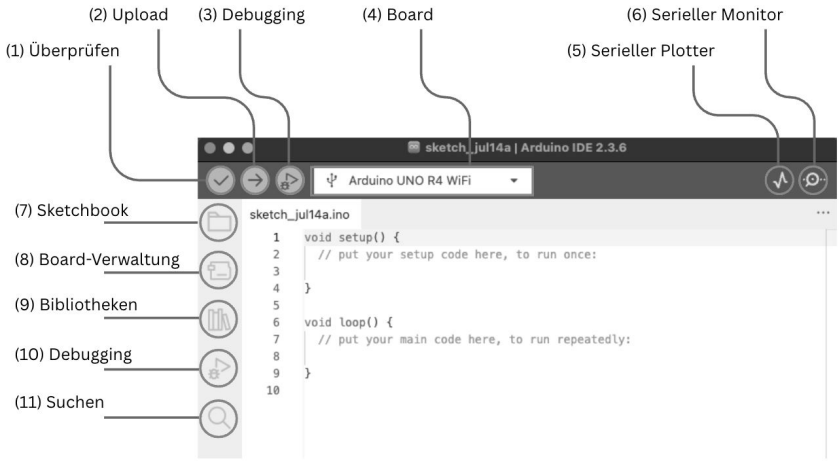
Lerne die IDE kennen

Wenn du die Arduino IDE zum allerersten Mal startest, passiert nichts Außergewöhnliches. Es gibt kein Tutorial und keine Tour durch das Programm. Deshalb übernehme ich das jetzt.

In diesem und den folgenden Abschnitten arbeite ich 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.



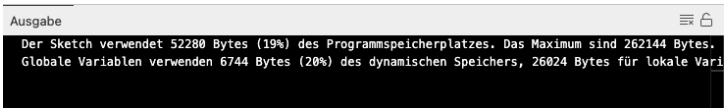
Von links nach rechts sind das:

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. **Upload:** Mit diesem Button lädst du deinen Sketch auf den Arduino hoch. Vor dem Upload findet auch immer zunächst eine Überprüfung des Codes statt.
3. **Debugging:** Hiermit kannst du dich mit Hardware-Unterstützung des angeschlossenen Boards auf Fehlersuche begeben. Allerdings ist dieses Feature nicht für den Arduino UNO verfügbar, sondern Boards vor allem der MKR-Reihe vorbehalten.
4. **Board:** Über dieses Menü wählst du den Microcontroller und den Port aus, an dem du ihn am Computer angeschlossen hast.
5. **Serieller Plotter:** Der Plotter stellt dir z.B. Sensordaten als Graphen dar, damit du sie über einen längeren Zeitraum auswerten kannst.
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 genauer.
7. **Sketchbook:** Hierüber kannst du deine gespeicherten Programme laden – das Sketchbook funktioniert so ähnlich wie der Datei-Explorer oder Finder auf dem PC oder Mac.
8. **Board-Verwaltung:** Hier kannst du Microcontroller wie den UNO R4 oder den ESP32 in der IDE installieren, damit du sie hierüber programmieren kannst.
9. **Bibliotheken:** Dahinter verbirgt sich ein Verzeichnis aller verfügbaren Bibliotheken, die du mit einem Klick installieren kannst. Hierzu später mehr.
10. **Debugging:** siehe oben.

11. **Suchen:** Hiermit kannst du nach Stellen im Code suchen. Das geht aber auch einfacher über die Tastenkombination Strg+F.

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 fehlendes Semikolon am Ende einer Zeile.

Der Button **Fehlermeldungen kopieren** ist oft praktisch. Damit kopierst du die Meldungen in die Zwischenablage und kannst sie gleich darauf in einer Suchmaschine verwenden. Du wirst sehen, **Suchmaschinen (und heute auch oft künstliche Intelligenz) sind oft der beste Freund beim Programmieren** – nicht nur als Anfänger.

Bevor es losgeht: Den UNO R4 in der IDE installieren

Damit du den Arduino UNO R4 gleich bequem mit der Arduino IDE programmieren kannst, sind zunächst ein paar Handgriffe nötig. **Falls du einen UNO R3 besitzt, musst du nichts tun** – dieses Board ist bereits in der Arduino IDE installiert.

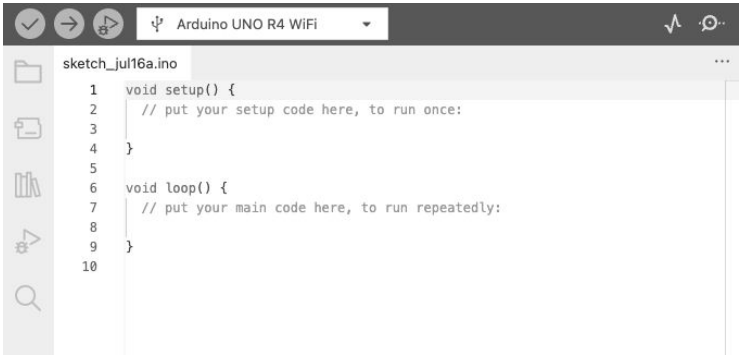
Klicke im Menü links auf das Icon der Board-Verwaltung und trage im Suchfeld oben **UNO R4** ein. Nun siehst du das Paket **Arduino UNO R4 Boards**, das du mit einem Klick auf **Installieren** bereitstellst. Die aktuelle Version (hier 1.5.0, aber diese Versionsnummer kann bei dir schon höher sein) über dem Button kannst du eingestellt lassen.

Und das war es schon, nun kannst du deinen UNO R4 im entsprechenden Menü (Punkt 4 in der Liste oben) auswählen. Im nächsten Kapitel wenden wir uns dem Code zu – und du wirst deinen ersten Sketch schreiben.

Jetzt wird es Zeit für die Praxis! In den folgenden Abschnitten 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. Hierfür 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**.

Der Loop

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 deines 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 **Neuer Sketch**. 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 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.

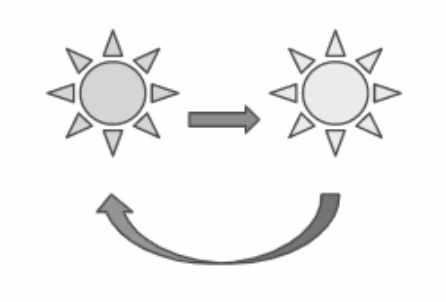
Hinweis: Achte darauf, dass du jede Zeile, die nicht mit einer geschweiften Klammer endet, mit einem Semikolon **;** abschließt. Ansonsten kann dein Sketch nicht hochgeladen werden. Von dieser Regel gibt es Ausnahmen, die jedoch jetzt noch nicht wichtig sind.

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

Der Loop

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.

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

Dein Arduino ist 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 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 allein.**

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! Trage den folgenden Sketch in deine Arduino IDE ein 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 sogar etwas morsen, indem du die LED unterschiedlich lang aufleuchten lässt.

Noch ein kurzer Seitenblick auf die Kommentare im Sketch. In C++ leitest du diese mit zwei Schrägstrichen **//** ein. Dahinter kannst du dir dann Notizen machen, die z.B. erläutern, was in diesem Teil des Codes passiert – oder wie hier, wozu die Funktionen verwendet werden.

```
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);
}
```