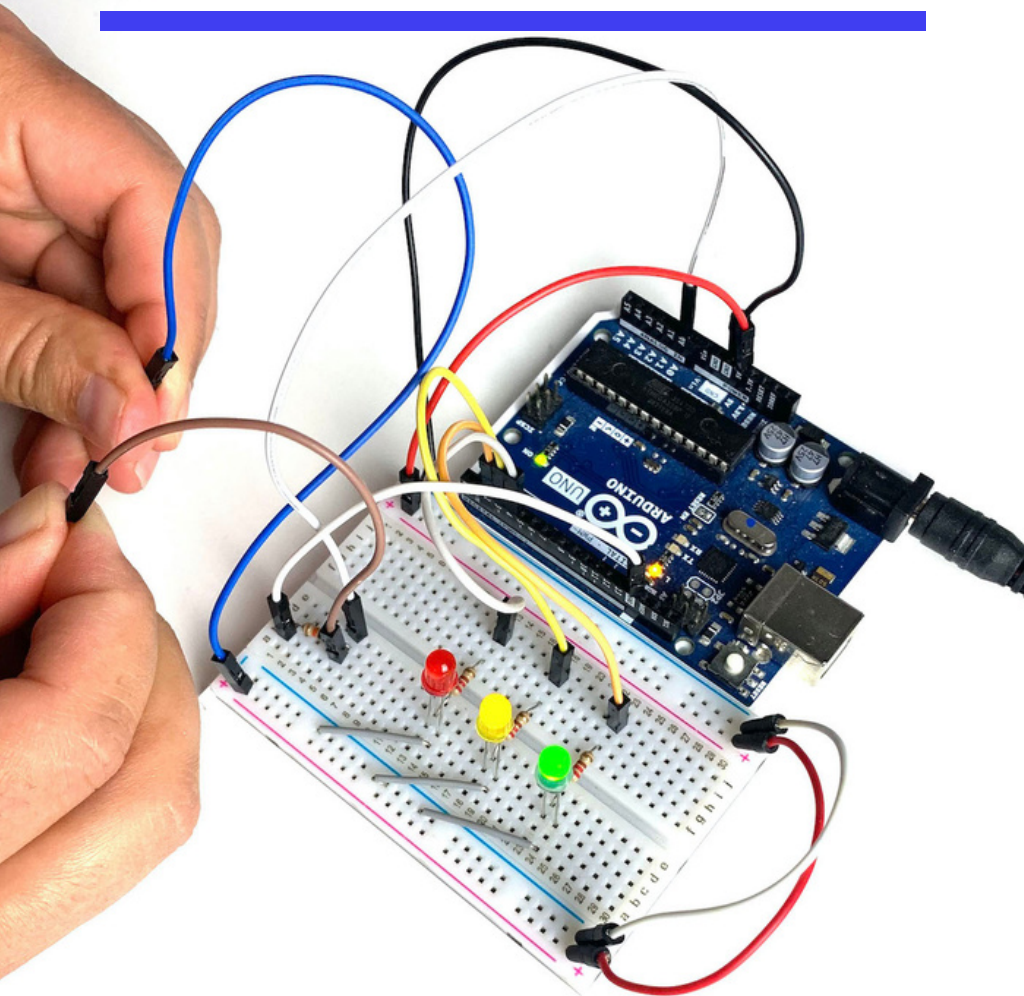


LEG' LOS MIT DEM ARDUINO



01

Der Arduino UNO

02

Programmieren mit der
Arduino IDE

03

Dein erster Sketch

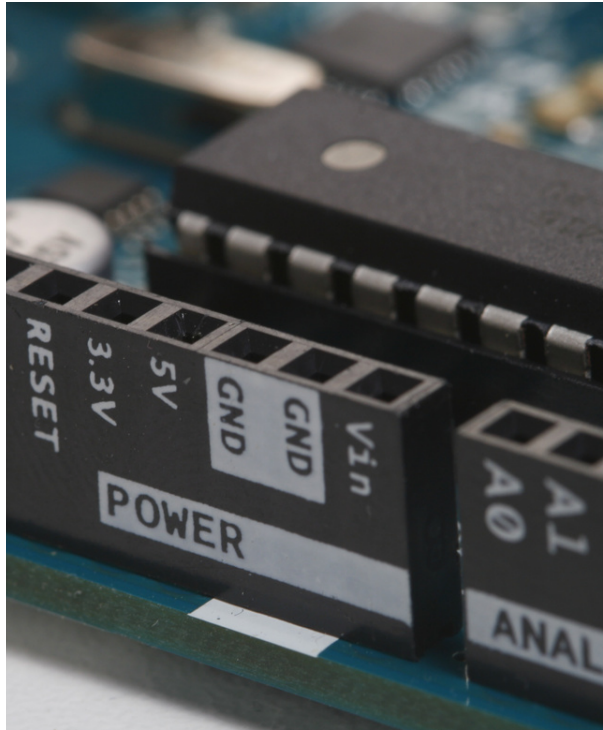
04

Baue deinen eigenen
Lügendetektor

INHALT



01 DER ARDUINO UNO



Der erste Arduino Microcontroller ist auch gleichzeitig der bekannteste: der 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.

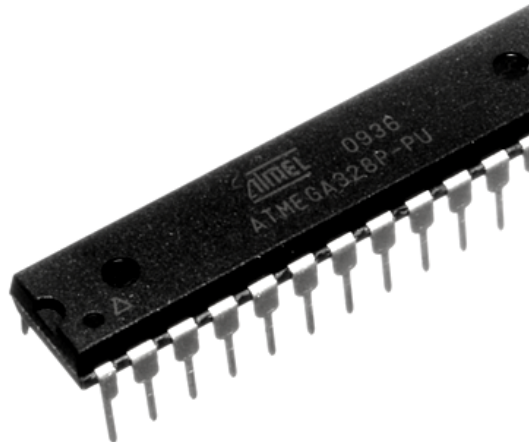
Und das mit Erfolg. Im Jahr 2005 erblickte der Arduino UNO das Licht der Welt – und es dauerte nicht lange, bis er die Herzen der Maker rund um den Globus erobert hatte.

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.



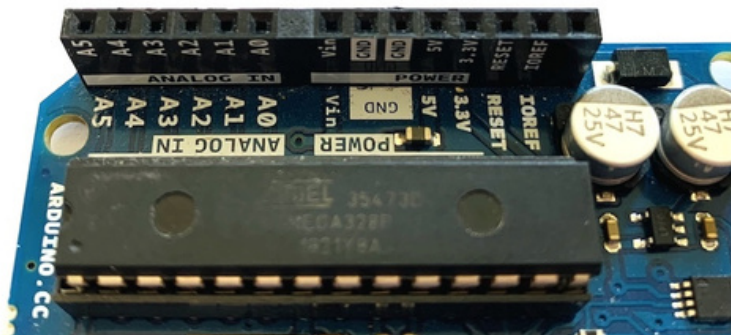
Auf diesem Chip landen deine Programme, die du hochlädst – wofür dir 32KB zur Verfügung stehen. Gemessen an modernen Speichermedien klingt das nach nichts, du wirst aber sehen, dass du damit schon unheimlich viel anfangen kannst.

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.

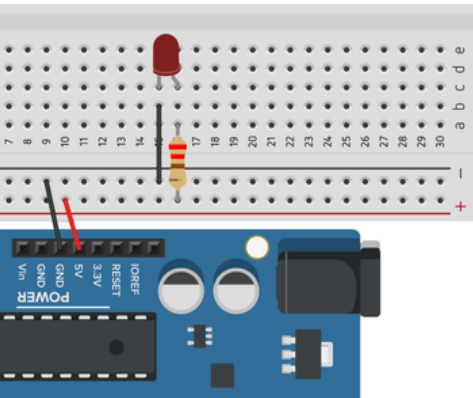
Wenn du hier also zum Beispiel eine LED in die Buchse A5 steckst, verbindest du sie dadurch mit dem entsprechenden Pin am Microchip.

DIE WICHTIGSTEN PINS DES ARDUINO UNO

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



Hier haben wir zunächst Pins, die mit Power gekennzeichnet sind. Die beiden Pins **GND** dienen als "Minuspol" oder "Erde". Das heißt, du kannst hier beispielsweise die Kathode (kurzes Bein) einer LED anschließen, um den Stromkreis zu schließen, in dem sie sich befindet.



In der Skizze oben siehst du, dass die Anode (langes Bein) der LED über einen Widerstand 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 allermeisten Hobby-Bauteile wie Servo-Motoren, Sensoren etc. benötigen eine dieser beiden Spannungen.

Im Bild oben siehst du den Sensor TMP36 am Arduino UNO. Durch Anpassen der Temperatur verändern sich die Zahlen rechts. Die Analog-Pins sind also immer dann nützlich, wenn du ein veränderbares Signal messen möchtest – und nicht nur 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 diesen Pins kannst du die Signale Null (Aus) und Eins (An) messen und ausgeben. Das bedeutet z.B., dass du eine LED über einen der Pins ein- 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.

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 sein Code fängt von vorne an.

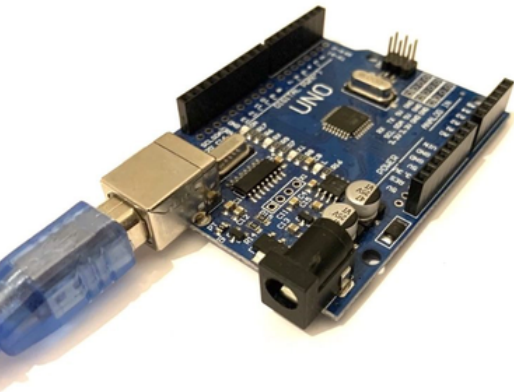
Und das soll es vorerst gewesen sein. Du kennst nun die wichtigsten Komponenten deines Arduino UNO, die du immer wieder verwenden wirst.

DIE STROMVERSORGUNG

Damit dein Arduino UNO seine Arbeit aufnehmen kann, benötigt er vor allem eines: Strom. Hierbei hast du mehrere Möglichkeiten, ihn mit Strom zu versorgen.

ÜBER EIN USB-KABEL

Am häufigsten wirst du ihn von deinem Computer aus über ein USB-Kabel mit Strom versorgen. Da du deine Programme ebenfalls per USB an den Arduino überträgst, ist das die einfachste und eine naheliegende Möglichkeit. Hiermit versorgst du deinen Microcontroller mit 5 Volt.

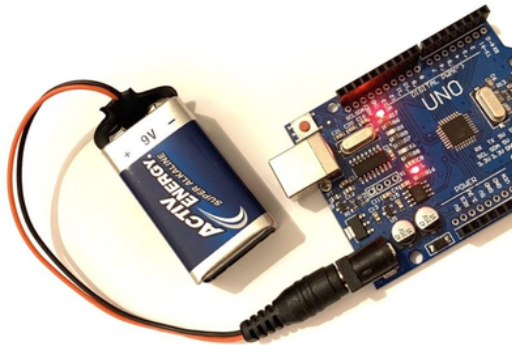


Achte darauf, dass du ein USB-Kabel verwendest, das auch Daten übertragen kann. Eine häufige Fehlerquelle für fehlgeschlagene Programm-Uploads sind tatsächlich Kabel, die nur Strom leiten – aber keine Daten.

MIT EINER 9V-BATTERIE

Früher oder später wirst du Projekte bauen, die du gerne unabhängig von deinem Computer betreiben möchtest. Da dein Arduino UNO den Computer nicht mehr braucht, sobald das Programm auf ihm gespeichert ist, kannst du auch eine andere Stromquelle verwenden.

Hierfür hat der Arduino UNO noch einen weiteren Eingang für eine 9-Volt-Batterie. Alles was du benötigst, ist also eine Batterie und ein passendes Kabel. Dieses findest du in den meisten Starter Kits oder für ein paar Cent im Handel.



Als mobile Stromquelle kann dir natürlich auch eine Powerbank dienen. Hierbei würde dann wiederum das USB-Kabel zum Einsatz kommen.

02 PROGRAMMIEREN MIT DER ARDUINO IDE



Für Einsteiger ist die Arduino IDE (Integrated Development Environment) meist die erste Wahl – und das zu Recht. Du kannst alle Arduino-Boards mit ihr programmieren und 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 Anfang des eigentlich recht komplexen Themas Hardware-Programmierung leicht.

In den folgenden Lektionen schauen wir uns zunächst an, wie du die Arduino IDE installierst. Anschließend folgt eine kleine Tour durch die wichtigsten Funktionen, um mit dem ersten Projekt 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 Webseite arduino.cc](https://www.arduino.cc) findest du die jeweils aktuelle Version. Ein Klick auf dein Betriebssystem startet den Download.

DOWNLOAD OPTIONS

Windows Win 7 and newer

Windows ZIP file

Windows app Win 8.1 or 10



Linux 32 bits

Linux 64 bits

Linux ARM 32 bits

Linux ARM 64 bits

Mac OS X 10.10 or newer

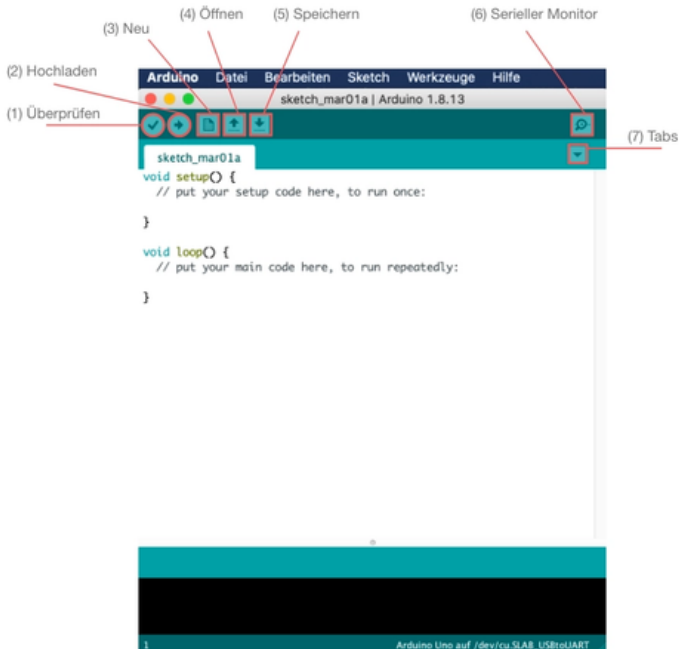
Sobald die Datei auf deinem Rechner ist, starte sie (bzw. öffne die ZIP-Datei) und folge den weiteren Anweisungen. Anschließend kannst du die Arduino IDE öffnen.

LERNE DIE IDE KENNEN

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

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.



(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 Fenster 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 mehr.

(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.



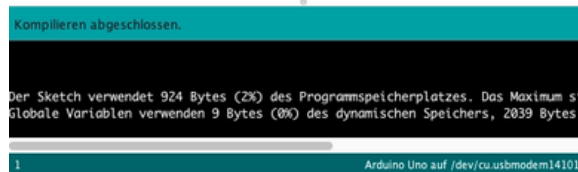
Wenn bei dir hier noch nicht **Arduino Uno** steht, klicke auf den Menüpunkt und wähle zunächst **Arduino AVR Boards**. Jetzt öffnet sich ein weiteres Untermenü, in dem du dann den **Arduino Uno** auswählen kannst.

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

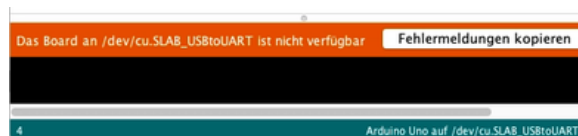
Ebenfalls im Menü Werkzeuge findest du den Eintrag **Port**. Wähle hier den USB-Port, an dem dein Arduino angeschlossen ist. Das ist zunächst alles, was du machen musst, um deinen Arduino mit der IDE programmieren zu können.

INFORMATIONEN UND FEHLERMELDUNGEN

Zuletzt schauen wir noch auf den unteren Teil 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, sieht du etwas in dieser Art:



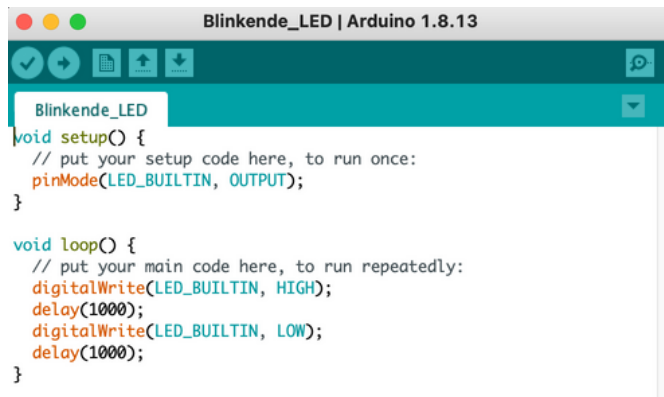
Eine Fehlermeldung sieht hingegen beispielsweise folgendermaßen aus:



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 für Anfänger.

03 DEIN ERSTER SKETCH



```
Blinkende_LED | Arduino 1.8.13
Blinkende_LED
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);
}
```

Jetzt wird es Zeit für die Praxis!

Auf den folgenden Seiten lernst du das Grundgerüst eines Arduino-Sketchs kennen und schreibst dein erstes eigenes Programm.

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

The image shows a screenshot of the Arduino IDE interface. At the top, the window title is "sketch_mar01c | Arduino 1.8.13". Below the title bar is a dark teal toolbar with icons for file operations and a search icon. The main editor area has a teal header with the filename "sketch_mar01c" and a dropdown arrow. The code in the editor is as follows:

```
void setup() {  
  // put your setup code here, to run once:  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

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.

Bald 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 DEINES ARDUINOS BLINKEN

Zeit für dein erstes Programm! Auf deinem Arduino UNO befinden sich mehrere kleine LEDs – eine hiervon wirst du nun immer wieder an- 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 eines 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.

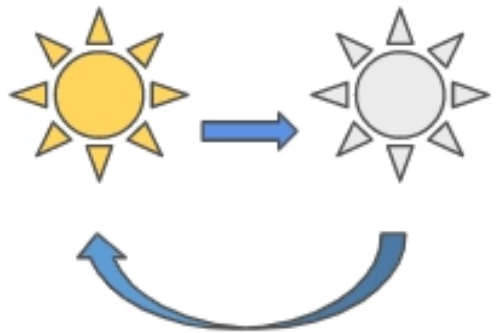
Der zweite Parameter bestimmt dann die Richtung – entweder **OUTPUT** oder **INPUT**. Wenn du eine LED steuern möchtest, muss der Arduino ihr entsprechende Signale senden. Vom Arduino aus gesehen ist das also ein **OUTPUT**. Wenn jedoch zum Beispiel ein Sensor Messdaten an den Arduino sendet, ist das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

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. Im vorangegangenen Abschnitt hast du gelernt, dass der Code innerhalb des Loops sich ständig wiederholt. Das können wir uns für das Blinken der LED zunutze machen.

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 soviel 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 anzuschalten. Aber das ist natürlich noch kein ordentliches Blinken.

WARTE KURZ

So ein Arduino ist erstaunlich schnell. Würdest du die LED anschalten und sie sofort danach wieder ausschalten, würdest du vom Blinken nichts mitbekommen. Der Wechsel von Ein 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 - ja, richtig - Funktion **delay()**. Dieser Befehl 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.

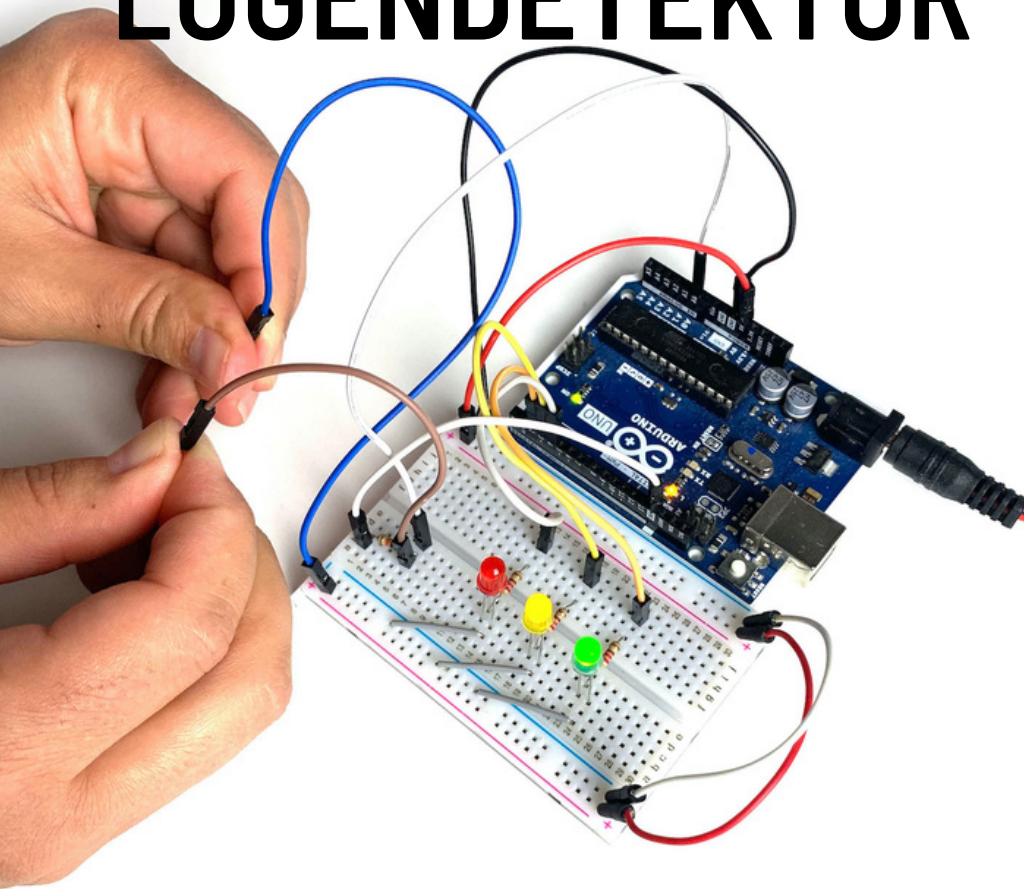
Sobald dein Programm das zweite Mal `delay(1000);` abgearbeitet hat, ist es am Ende des Loops angekommen und springt wieder zu dessen Anfang - also in die Zeile `digitalWrite(LED_BUILTIN, HIGH);`

Auf der nächsten Seite findest du den gesamten Sketch zum Herauskopieren. Lade ihn auf deinen Arduino und spiele anschließend 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.

SKETCH BLINKENDE LED

```
void setup() {  
  pinMode(LED_BUILTIN, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(LED_BUILTIN, HIGH);  
  delay(1000);  
  digitalWrite(LED_BUILTIN, LOW);  
  delay(1000);  
}
```

03 BAUE DEINEN EIGENEN LÜGENDETEKTOR



Du bist dir nicht ganz sicher, ob du gerade angeflunkert wirst? Dann wird es Zeit für deinen eigenen kleinen Lügendetektor.

Deine Testperson muss hierfür nur zwei Kabel in die Hand nehmen und deine Fragen beantworten – und schon weißt du, ob sie lügt oder nicht.

Mit diesem Projekt wirst du natürlich nicht wirklich herausfinden, ob du angelogen wirst. Schließlich sind auch "professionelle" Lügendetektoren nicht fähig, die Wahrheit zuverlässig ans Tageslicht zu bringen. Aber für einen kleinen Spaß unter Freunden reicht es allemal!

SO FUNKTIONIERT DER LÜGENDETEKTOR

Dieses Projekt ist nicht kompliziert: Du verbindest drei farbige LEDs mit dem Arduino, die dir anzeigen, wie "nah an der Wahrheit" eine Antwort ist. Ein weiterer Stromkreis wird geschlossen, indem deine Testperson zwei Kabel in die Hand nimmt. Eines dieser Kabel verbindest du mit 5V. Das andere besteht genau genommen aus zwei Kabeln, zwischen denen du einen 10k Ω Widerstand steckst, den du mit GND verbindest.

Dieser Widerstand ist ein sogenannter Pulldown-Widerstand, der den Messwert an Pin A0 auf Null "herunterzieht". Würdest du hier keinen Widerstand verwenden, würden die Messwerte "verrücktspielen" – du würdest hier alle möglichen Werte empfangen.

Wie das aussieht, kannst du prüfen, indem du später den Widerstand probeweise entfernst.

Je nachdem, wie sehr die Person an den Fingern schwitzt, desto geringer ist ihr Hautwiderstand. Diese kleinen Unterschiede kannst du messen und in deinem Seriellen Monitor nachverfolgen.

Nehmen wir an, die Testperson fängt beim Lügen an zu schwitzen: Der Widerstand der Haut verringert sich und sie wird leitfähiger – der Messwert verändert sich und steigt. Bleiben die Finger trocken, so ist der fließende Strom einem größeren Widerstand ausgesetzt – und der Messwert bleibt niedrig.

DER AUFBAU

Orientiere dich beim Aufbau des Lügendetektors an folgender Skizze. Für die drei LEDs benötigst du noch jeweils einen Widerstand von 220Ω . Diese verhindern, dass die LEDs durchbrennen, da sie die zugeführte Strommenge regulieren.

DER SKETCH

Einiges im folgenden Programm hast du bereits kennengelernt, z.B. die Setup- und Loop-Funktionen und den Befehl `digitalWrite()`.

Für den Lügendetektor benötigst du jedoch etwas mehr, wie z.B. **Variablen**, bedingte Anweisungen mit `if` und `else` und den **Seriellen Monitor**. Auf den folgenden Seiten reißen wir diese Themen an. Wenn du mehr darüber wissen möchtest, empfehlen wir dir unseren [Arduino Online-Kurs](#). Hier lernst du wichtige Grundlagen von C++ und der Elektronik und baust noch viele weitere spannende Projekte.

VARIABLEN

In einer Variablen kannst du Werte, wie z.B. Zahlen speichern. In unserem Sketch sind diese Zahlen die Anschluss-Pins der drei LEDs. Der Vorteil einer Variablen ist, dass du sie nur einmal zu Beginn des Sketchs definierst und dann immer wieder verwenden und aktualisieren kannst.

Variablen, die ganze Zahlen beinhalten, haben in C++ den Typ `int` (für engl. integer). Weise ihnen die Pins des Arduinos wie folgt zu:

```
int red = 4;  
int yellow = 3;  
int green = 2;
```

DER SERIELLE MONITOR

Vom Seriellen Monitor hast du bereits im Abschnitt über die Arduino IDE etwas gehört. Hier kannst du dir z.B. Werte anzeigen lassen, die dein Arduino ausgibt.

Damit der Monitor verfügbar ist, musst du ihn zunächst in der Setup-Funktion "starten":

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

Später im Loop gibst du dann die Werte mit dem Befehl `Serial.println()` aus – doch dazu gleich mehr.

Zunächst benötigst du noch die `pinModes` für die LEDs mit der Richtung `OUTPUT`.

```
pinMode(green, OUTPUT);
pinMode(yellow, OUTPUT);
pinMode(red, OUTPUT);
}
```

DIE LOOP-FUNKTION

Du kennst ja bereits die Funktion `digitalWrite()`, mit der du z.B. eine LED auf HIGH, also einschalten kannst. Das Gegenstück hierzu ist die Funktion `analogRead()`. Damit liest dein Arduino Werte ein – in unserem Fall Werte des Hautwiderstands.

Hierbei wird gleich noch ein weiteres wichtiges Konzept im Programmieren wichtig: bedingte Anweisungen.

IF ... ELSE

Mit diesen Anweisungen baust du Abzweigungen in dein Programm ein. Wenn die eine Bedingung (if) zutrifft: Tue dies. Trifft etwas anderes zu (else), dann tue jenes.

Im Sketch sieht das so aus:

```
if (analogRead(A0) > 10) {  
  digitalWrite(green, HIGH);  
}
```

Wenn also der Wert, den dein Arduino über `analogRead()` liest, über 10 liegt – dann schaltet er die grüne LED ein.

Liegt der Wert jedoch darunter, dann schaltet er die LED aus:

```
else {  
    digitalWrite(green, LOW);  
}
```

So geht es weiter mit der gelben LED. Liegt der Messwert über 20, wird sie eingeschaltet. Bei allen Werten von 20 und darunter, bleibt sie ausgeschaltet.

```
if (analogRead(A0) > 20) {  
    digitalWrite(yellow, HIGH);  
}  
else {  
    digitalWrite(yellow , LOW);  
}
```

Schnellt der Wert über 50, geht schließlich die rote LED an. Hier hat deine Testperson offensichtlich klatschnasse Finger vor Angst.

```
if (analogRead(A0) > 50) {  
    digitalWrite(red, HIGH);  
}  
else {  
    digitalWrite(red, LOW);  
}
```

Zwei Dinge fehlen noch. Du selbst brauchst einen Ort, an dem du die Werte ablesen kannst: den **Seriellen Monitor**. Hier gibst du die eingelesenen Werte wie folgt aus:

```
Serial.println(analogRead(A0));
```

Wie du schon weißt, öffnest du den Monitor über die Lupe oben rechts in deiner Arduino IDE.

Zuletzt kommt noch ein kleiner **Delay**, der ein paar Millisekunden zwischen die Messungen des Arduinos fügt. Das tust du, damit diese in einem festen Takt erfolgen.

```
delay(20);  
}
```

Auf der folgenden Seite findest du den gesamten Sketch. Lade ihn auf deinen Arduino und experimentiere etwas mit der Feuchtigkeit deiner Finger und den eingestellten Schwellenwerten (10, 20, 50) im Sketch.

SKETCH LÜGENDETEKTOR

```
int red = 4;
int yellow = 3;
int green = 2;

void setup() {
  Serial.begin(9600);
  pinMode(green, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(red, OUTPUT);
}

void loop() {

  if (analogRead(A0) > 10) {
    digitalWrite(green, HIGH);
  }
  else {
    digitalWrite(green, LOW);
  }

  if (analogRead(A0) > 20) {
    digitalWrite(yellow, HIGH);
  }
  else {
    digitalWrite(yellow , LOW);
  }

  if (analogRead(A0) > 50) {
    digitalWrite(red, HIGH);
  }
  else {
    digitalWrite(red, LOW);
  }

  Serial.println(analogRead(A0));
  delay(20);
}
```


WIE GEHT ES WEITER?

Du hast es geschafft – dein eigener Arduino Lügendetektor steht vor dir.

Möchtest du mehr über C++ und Elektronik wissen? Weitere spannende Projekte mit Sensoren, Motoren und Displays bauen?

Dann wirf einen Blick auf unseren Arduino Online-Kurs! Mit diesem Code erhältst du beim Kauf 15% Rabatt:

ARDUINO15

[Zum Online-Kurs](#)

pollux labs