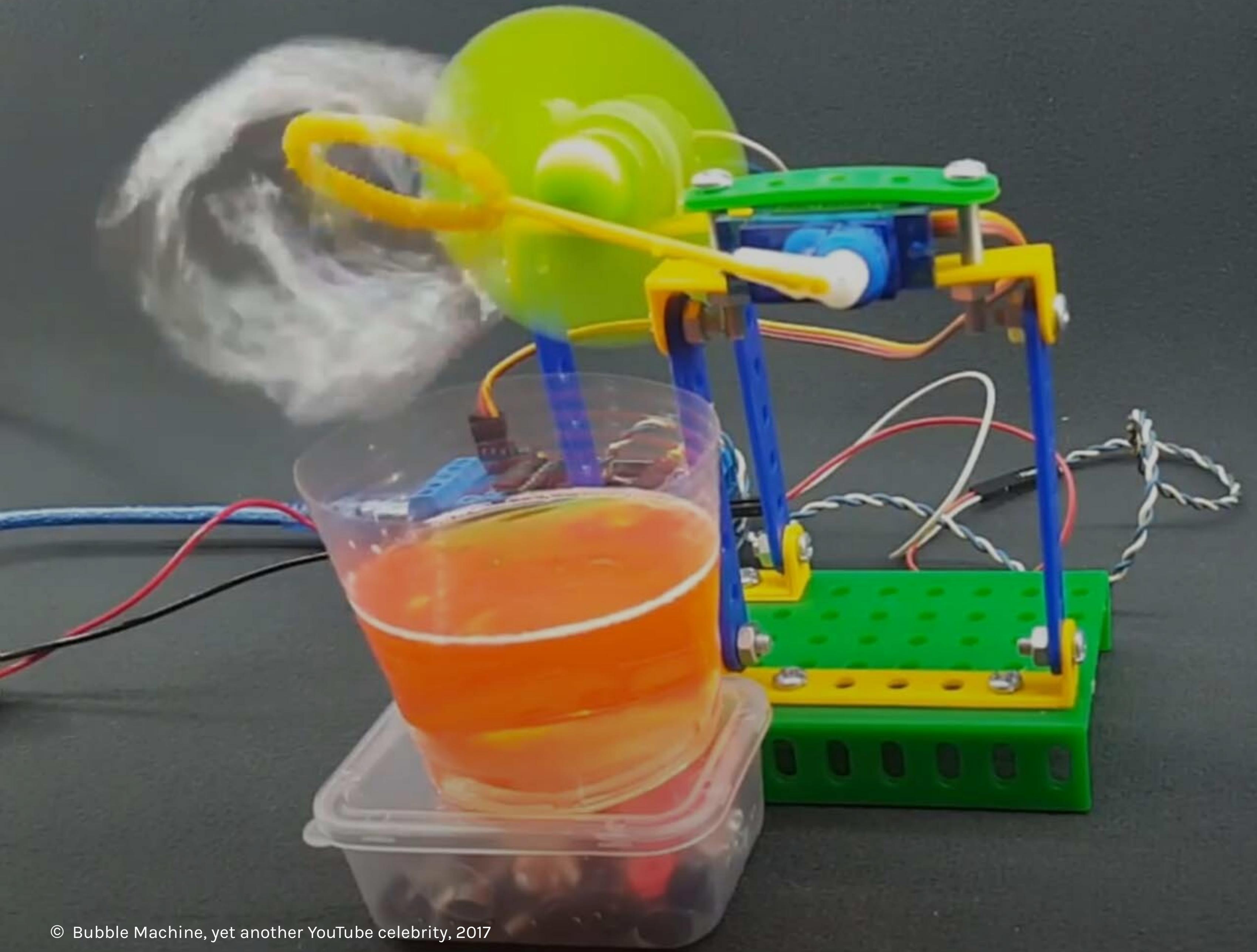


LOOK MUM
NO SCR
EIN

Look Mum No Screen

↪ I/O Eingabe und Ausgabe



Look Mum No Screen

↳ I/O Eingabe und Ausgabe



Look Mum No Screen

↳ I/O Eingabe und Ausgabe



Look Mum No Screen

↳ I/O Eingabe und Ausgabe



Digital In:

Es gibt zwei unterschiedliche Typen von Inputs. Digital und Analog.
Die Digitalen Inputs können lediglich zwei unterschiedliche Zustände abbilden:

HIGH oder **LOW**
1 oder **0**
5V oder **0V**

Digitale Inputs werden hauptsächlich dazu verwendet, um das Vorhandensein von Spannung zu erfassen.

Digitale Inputs können aber auch als Grundlage für zahlreiche digitale Kommunikationsprotokolle verwendet werden. Durch das Erzeugen eines 5V Impulses (**HIGH**) oder eines 0V Impulses (**LOW**) kann ein einfaches binäres Signal erzeugt werden. Dieses fungiert dann als Grundlage sämtlicher Datenverarbeitung.



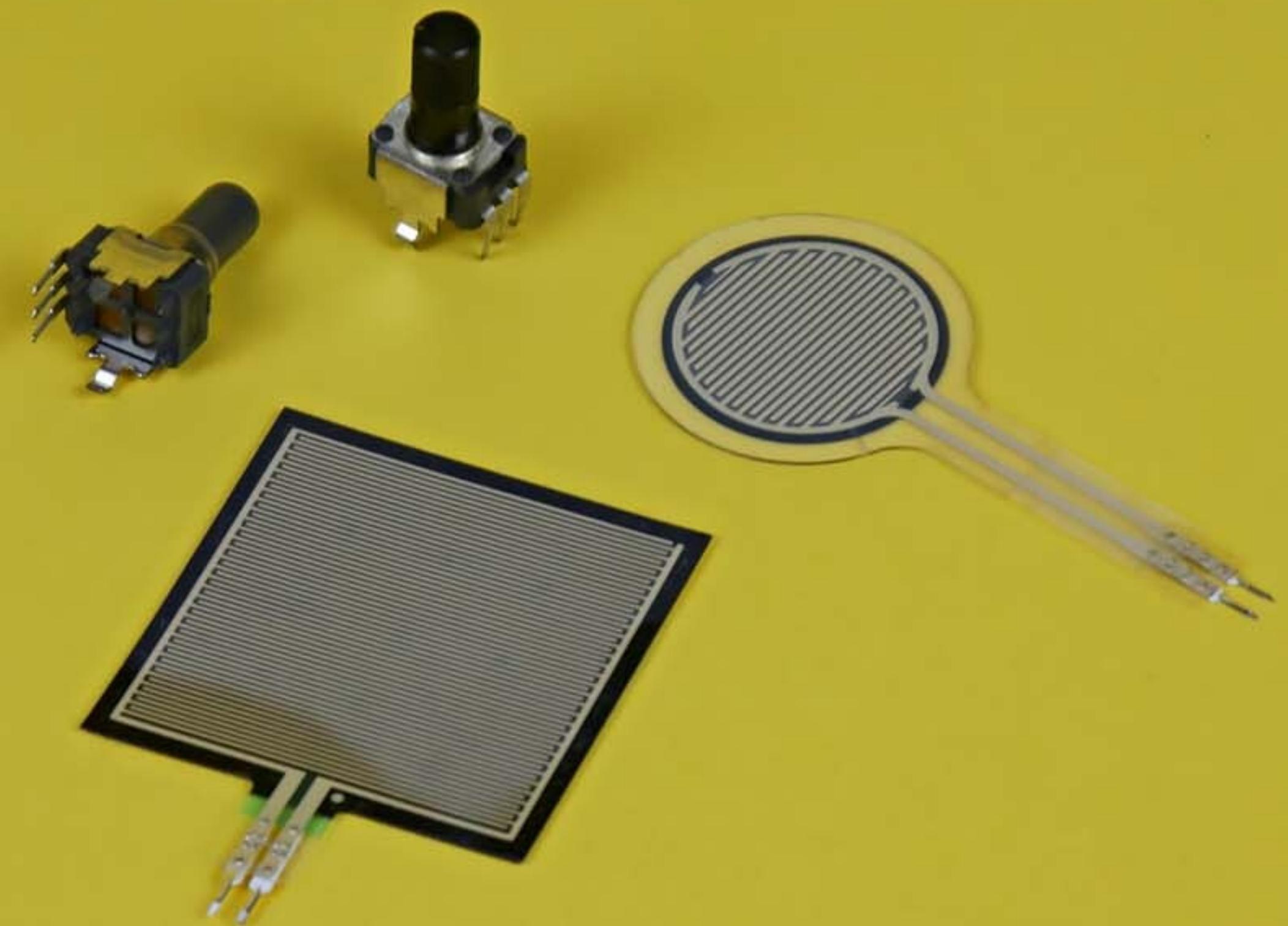
Analog In:

Die Analogen Inputs nehmen ein analoges Signal auf und führen dann eine 10-Bit-Analog-Zu-Digital Wandlung durch, um so eine Zahl zwischen **0** und **1023** zu erhalten.

Analoge Inputs eignen sich gut zum Auslesen von Widerstands-sensoren. Dabei handelt es sich dann um Sensoren, die dem Stromkreis einen Widerstand zuführen.

Look Mum No Screen

↪ I/O Eingabe und Ausgabe



Digital Out:

Ein digitaler Output kann entweder auf **HIGH (5V)** oder auf **LOW (0V)** gesetzt werden. Damit können dann Komponenten beispielsweise ein- oder ausgeschaltet werden.

Look Mum No Screen

↳ I/O Eingabe und Ausgabe



Analog Out:

Analog Out beschreibt eine eingebaute “Sonderfunktion”. Mit Hilfe der sog. Pulsweitenmodulation oder kurz **PWM** können wir einen analogähnlichen Output erzeugen.

PWM funktioniert indem wir den Output (Pin) schnell **HIGH** (5V) und schnell **LOW** (0V) schalten, um so ein analoges Signal zu simulieren.

Wenn wir beispielsweise eine LED ganz schnell ein- und ausschalten, erscheint die Helligkeit “gedimmt” und wir können verschiedene Helligkeitsstufen darstellen.

Look Mum No Screen

↳ I/O Eingabe und Ausgabe

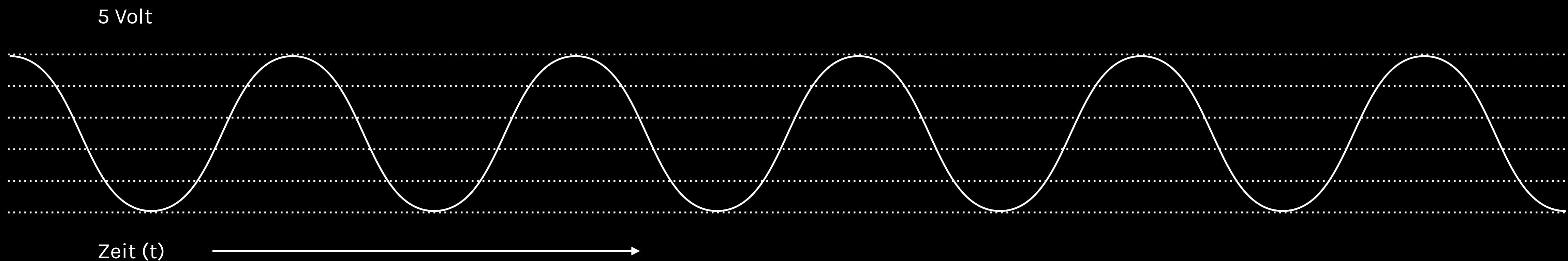


Analog und Digital: Im Bereich der Übertragungstechnik wird grundlegend zwischen der Information und dem Signal unterschieden. Die übertragene Information beschreibt das, was übertragen wird.

Das Signal hingegen definiert, wie es übertragen wird. Sowohl die Information als auch das Signal können analog oder digital sein.

Analoges Signal:

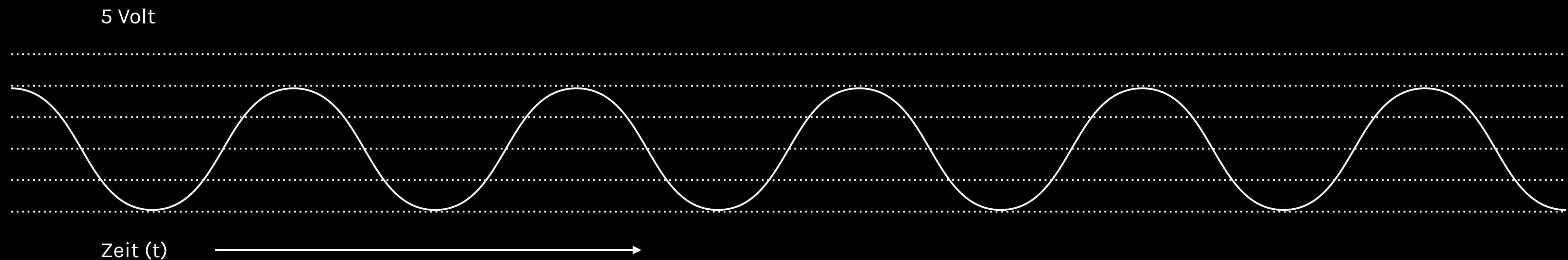
Ein analoges Signal ist eine physikalische Größe, die im Verlauf der Größe (Amplitude) als auch im zeitlichen Verlauf kontinuierliche Werte annehmen kann.



Analoges Signal:

Durch eine 10-Bit-Analog-Zu-Digital Wandlung liegt der zu erwartende Wertebereich zwischen:

0 – 1023 (ingesamt 1024 Werte).

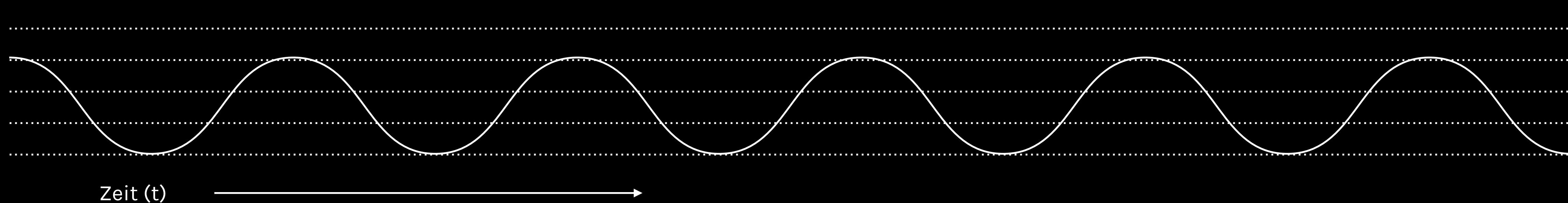


Analoges Signal:

Durch eine 10-Bit-Analog-Zu-Digital Wandlung liegt der zu erwartende Wertebereich zwischen:

0 – 1023 (ingesamt 1024 Werte).

5 Volt

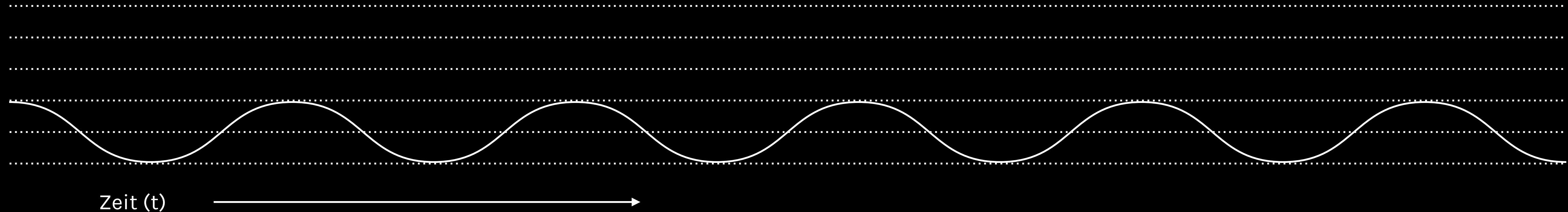


Analoges Signal:

Durch eine 10-Bit-Analog-Zu-Digital Wandlung liegt der zu erwartende Wertebereich zwischen:

0 – 1023 (ingesamt 1024 Werte).

5 Volt

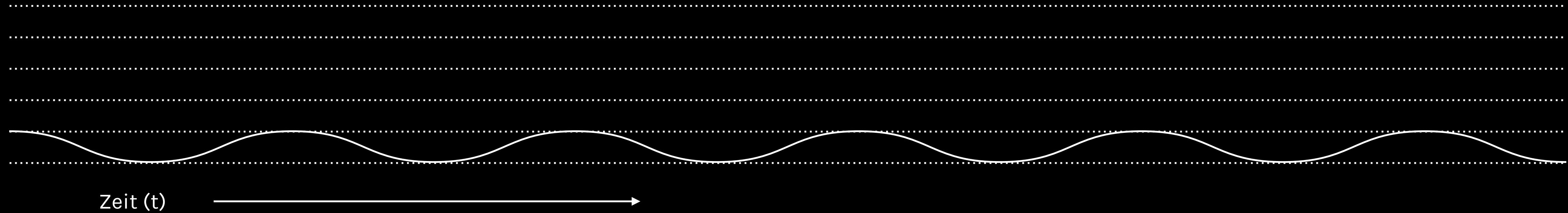


Analoges Signal:

Durch eine 10-Bit-Analog-Zu-Digital Wandlung liegt der zu erwartende Wertebereich zwischen:

0 – 1023 (ingesamt 1024 Werte).

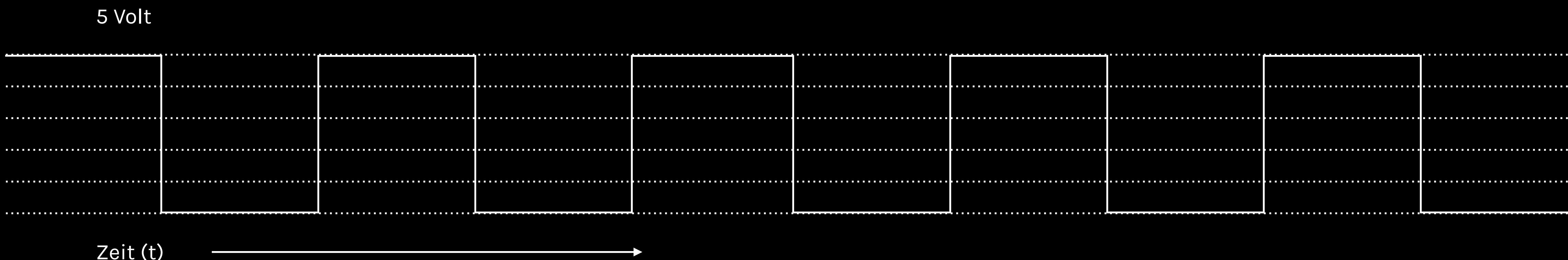
5 Volt



Digitale Signale:

Ein digitales Signal (ebenfalls eine physikalische Größe) kann nur bestimmte diskrete Werte annehmen. Die Werte entsprechen der Anzahl der vereinbarten Zustände. Werden zwei Zustände vereinbart, dann handelt es sich um binäre (digitale) Signale.

Der zu erwartende Wertebereich liegt hier zwischen: **0 – 1**.



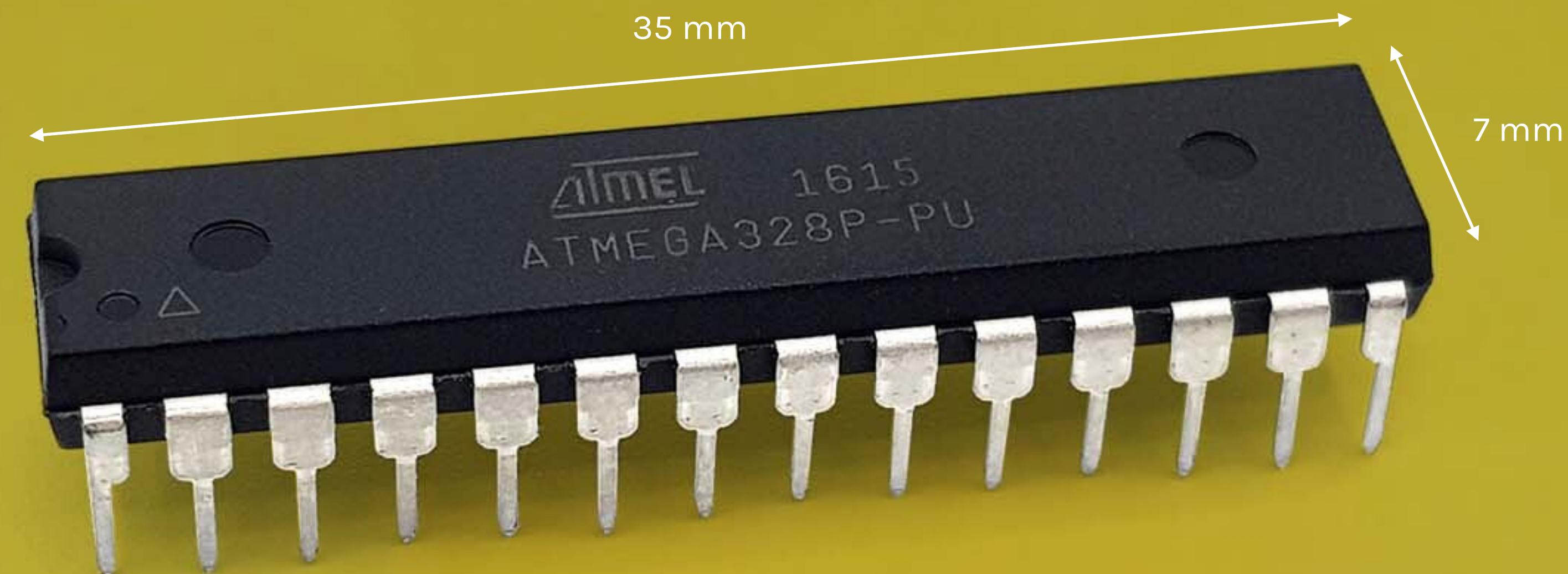
Look Mum No Screen

↪ I/O Eingabe und Ausgabe



Look Mum No Screen

↪ I/O Eingabe und Ausgabe



Mikrocontroller: Mikrocontroller werden auch als Halbleiterchips bezeichnet, die einen Prozessor und auch Peripheriefunktionen enthalten. Ebenfalls befindet sich in den meisten Fällen auch der Arbeitsspeicher teilweise oder auch komplett auf dem Chip.

Mikrocontroller: Mikrocontroller werden auch als Halbleiterchips bezeichnet, die einen Prozessor und auch Peripheriefunktionen enthalten. Ebenfalls befindet sich in den meisten Fällen auch der Arbeitsspeicher teilweise oder auch komplett auf dem Chip.

Entwicklerboard: Mit der Hilfe eines Entwicklerboards wird dem Endanwender die Nutzung des Mikrocontrollers erleichtert. Die meisten Entwicklerboards lassen eine direkte Verbindung verschiedener Sensoren oder Motoren durch ihre digitalen und analogen Ein- und Ausgänge zu.

Mit ihnen lassen sich einfache und meist temporäre Steckbrett Schaltungen umsetzen.

Look Mum No Screen

↳ I/O Eingabe und Ausgabe

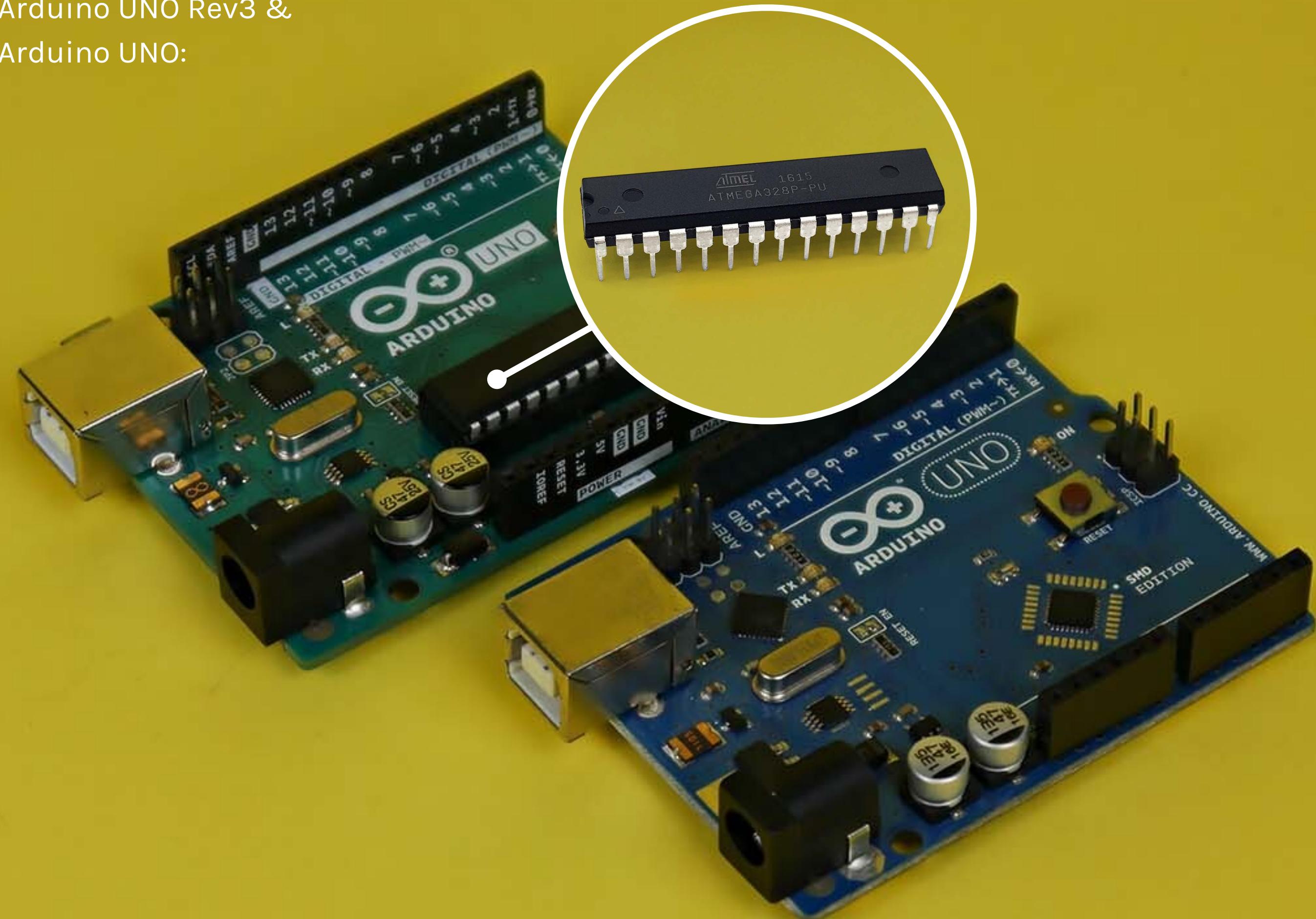
Arduino UNO Rev3 &
Arduino UNO:

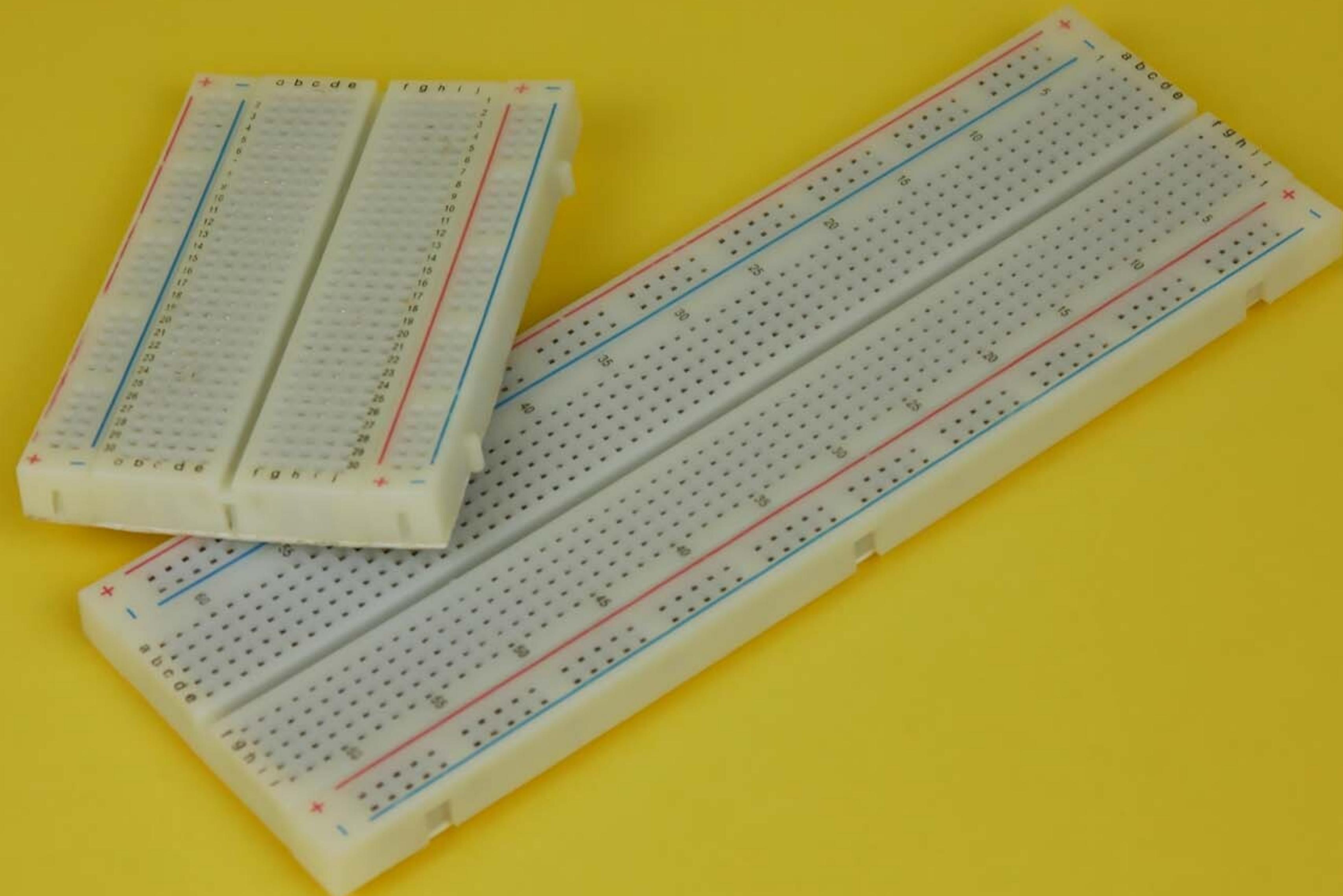


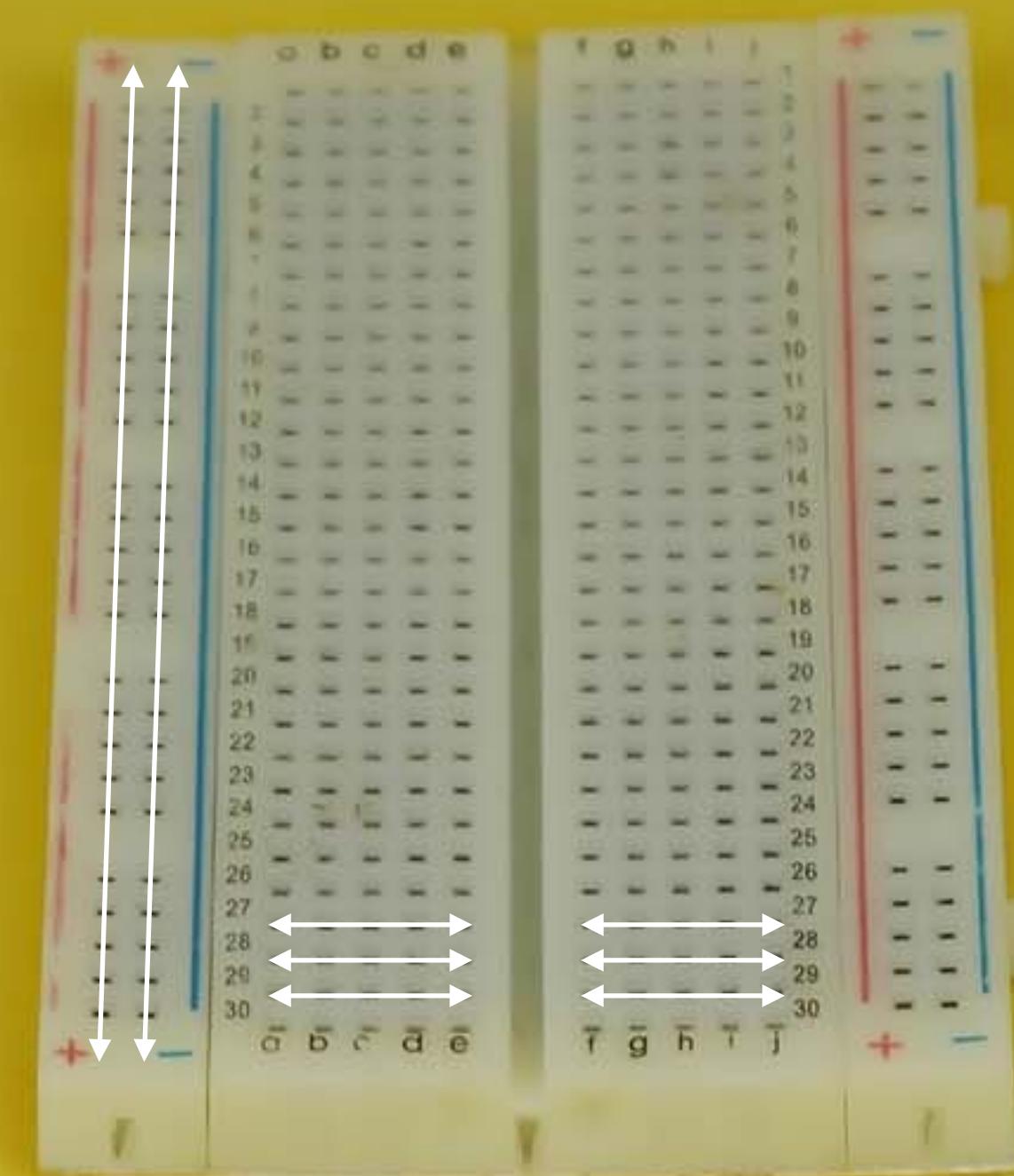
Look Mum No Screen

↳ I/O Eingabe und Ausgabe

Arduino UNO Rev3 &
Arduino UNO:







Bread Board:

Ein Breadboard oder lötfreies Steckbrett eignet sich hervorragend für die Herstellung von temporären Schaltkreisen und Prototypen.

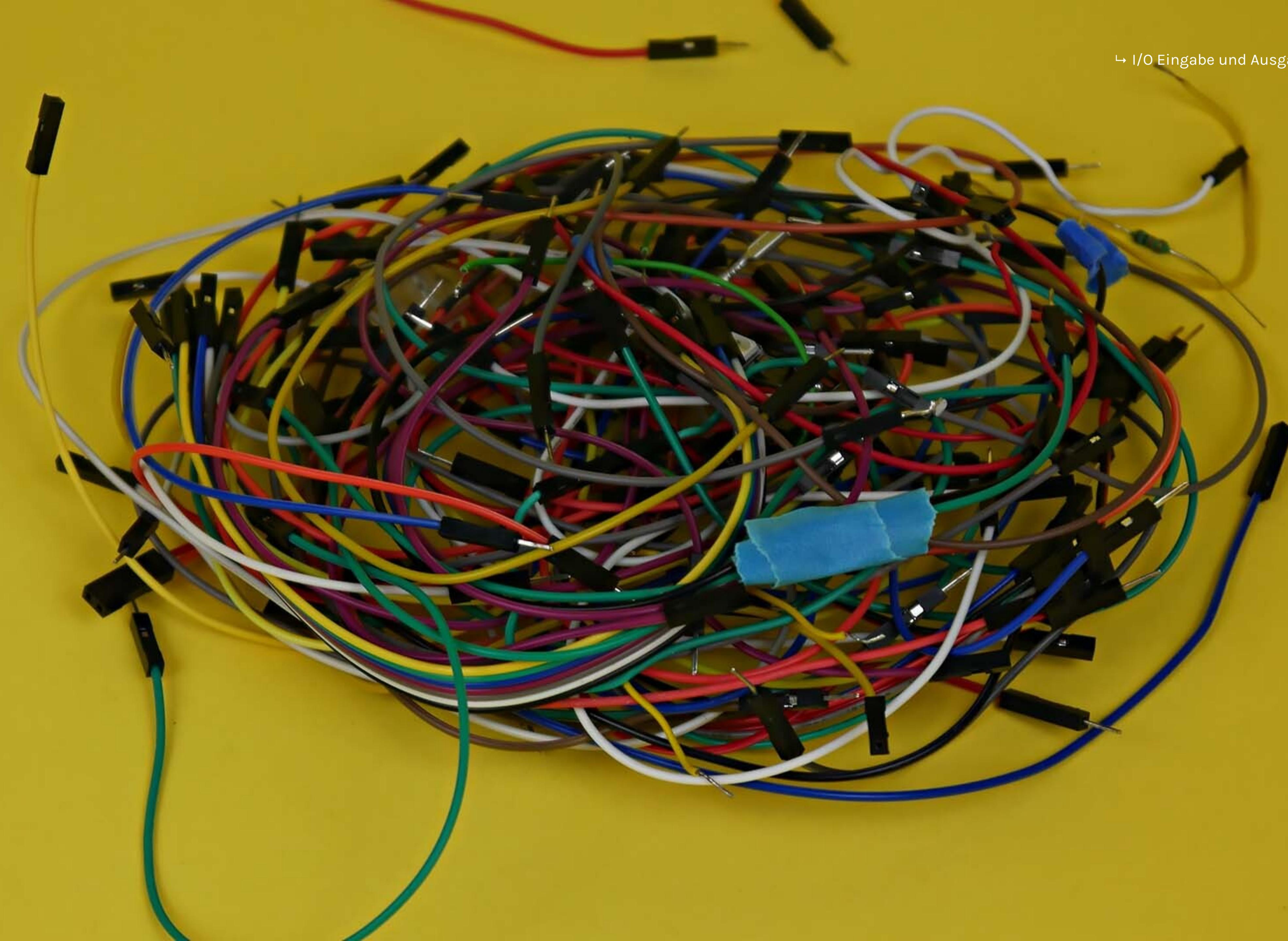


Widerstände:

Widerstände werden dazu verwendet, um die angelegte Energie (Spannung, Strom, Zeit) in Wärme umzuwandeln. Wir die Energie zu groß, wir das Bauteil im schlimmsten Fall zerstört.

Look Mum No Screen

↪ I/O Eingabe und Ausgabe



Drahtbrücken /
Jumper Wire:

Jumper Wire ermöglichen das schnelle und einfache Stecken aller Verbindungen für einen prototypischen Aufbau. Es gibt:

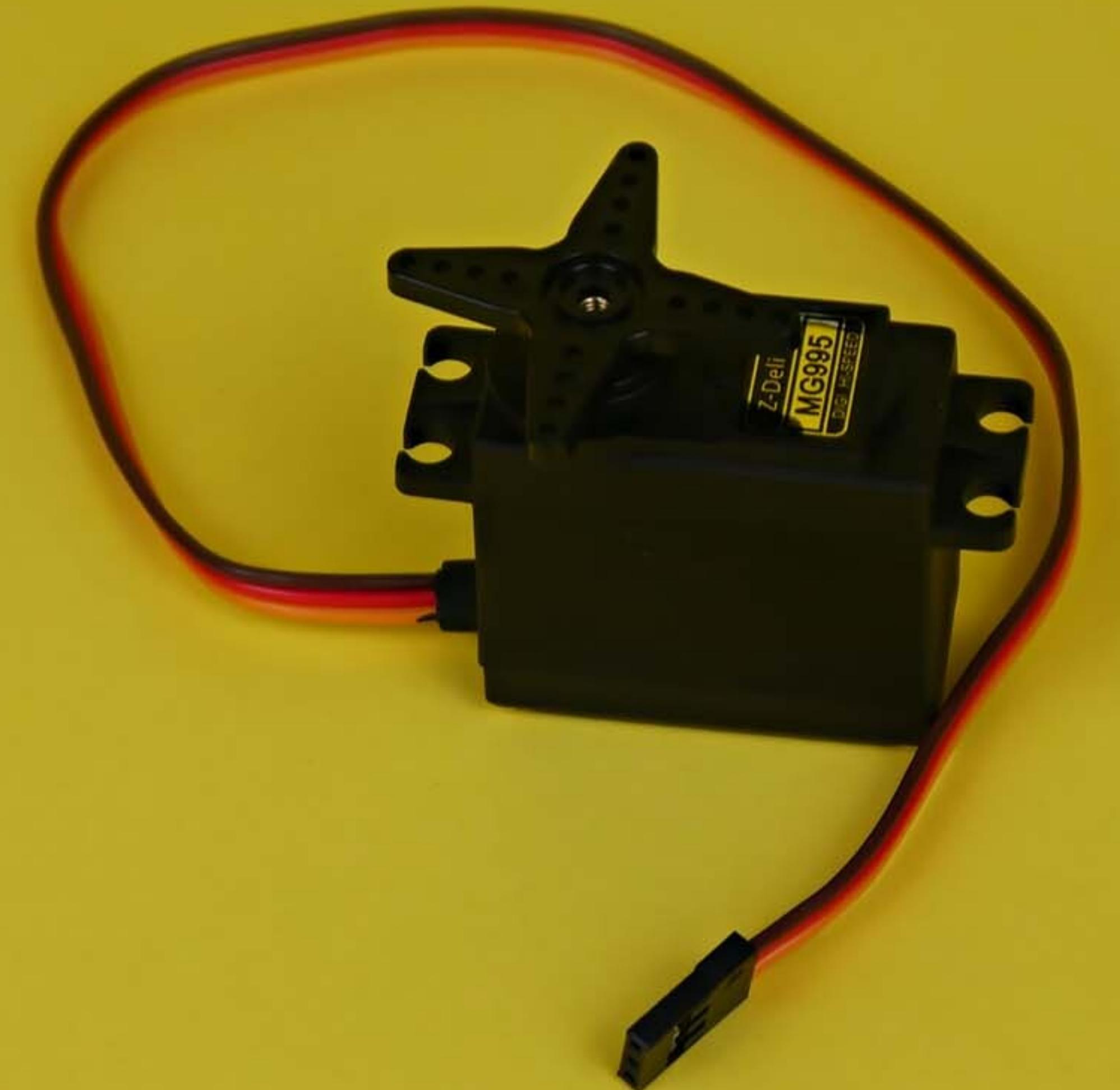
Male - Male

Male - Female

Female - Female

Look Mum No Screen

↳ I/O Eingabe und Ausgabe



Servomotor:

Als Servomotor wird ein Elektromotor bezeichnet, mit dem man die Kontrolle über Winkelposition, Drehgeschwindigkeit sowie Beschleunigung erlangt.

Look Mum No Screen

↳ I/O Eingabe und Ausgabe



Stepper Motor:

Mit einem Stepper Motor (Schrittmotor) können exakte und besonders kleine Schritte bzw. Ausrichtungen erlangt werden. Anders als bei einem Servomotor, kann der Stepper Motor volle 360° und mehr abbilden.



Elektrische Größen: Spannung / Volt (U)

Mit der elektrischen Spannung, auch als elektromotorische Kraft bezeichnet, wird eine Aussage darüber getroffen, wie stark der Antrieb des Stromes ist. Die Spannung ist also quasi die treibende Kraft, die dafür sorgt, dass der Strom fließt. Je höher die Spannung ist, desto größer ist der Fluss des elektrischen Stroms durch den Leiter bzw. Halbleiter.

Elektrische Größen: Spannung / Volt (U)

Mit der elektrischen Spannung, auch als elektromotorische Kraft bezeichnet, wird eine Aussage darüber getroffen, wie stark der Antrieb des Stromes ist. Die Spannung ist also quasi die treibende Kraft, die dafür sorgt, dass der Strom fließt. Je höher die Spannung ist, desto größer ist der Fluss des elektrischen Stroms durch den Leiter bzw. Halbleiter.

Stromstärke / Ampere (I)

Ampere ist die elektrische Basiseinheit für die Stromstärke. Ampere bezeichnet also die Menge an Elektronen beziehungsweise Ladungsträgern, die in einer bestimmten Zeitspanne durch eine Leitung fließen.

Daher finden sich Angaben zur Stromstärke auch meist an Sicherungskästen. Eine normale Haussicherung hat in der Regel eine Stromstärke von 16 Ampere.

Elektrische Größen: Leistung / Watt

Watt ist die Bezeichnung oder Elektrische Einheit für Leistung.

Dabei handelt es sich um den Energieumsatz pro Zeit. Demnach stellt die Einheit Watt dar, wie viel Strom entweder erzeugt oder verbraucht wird.

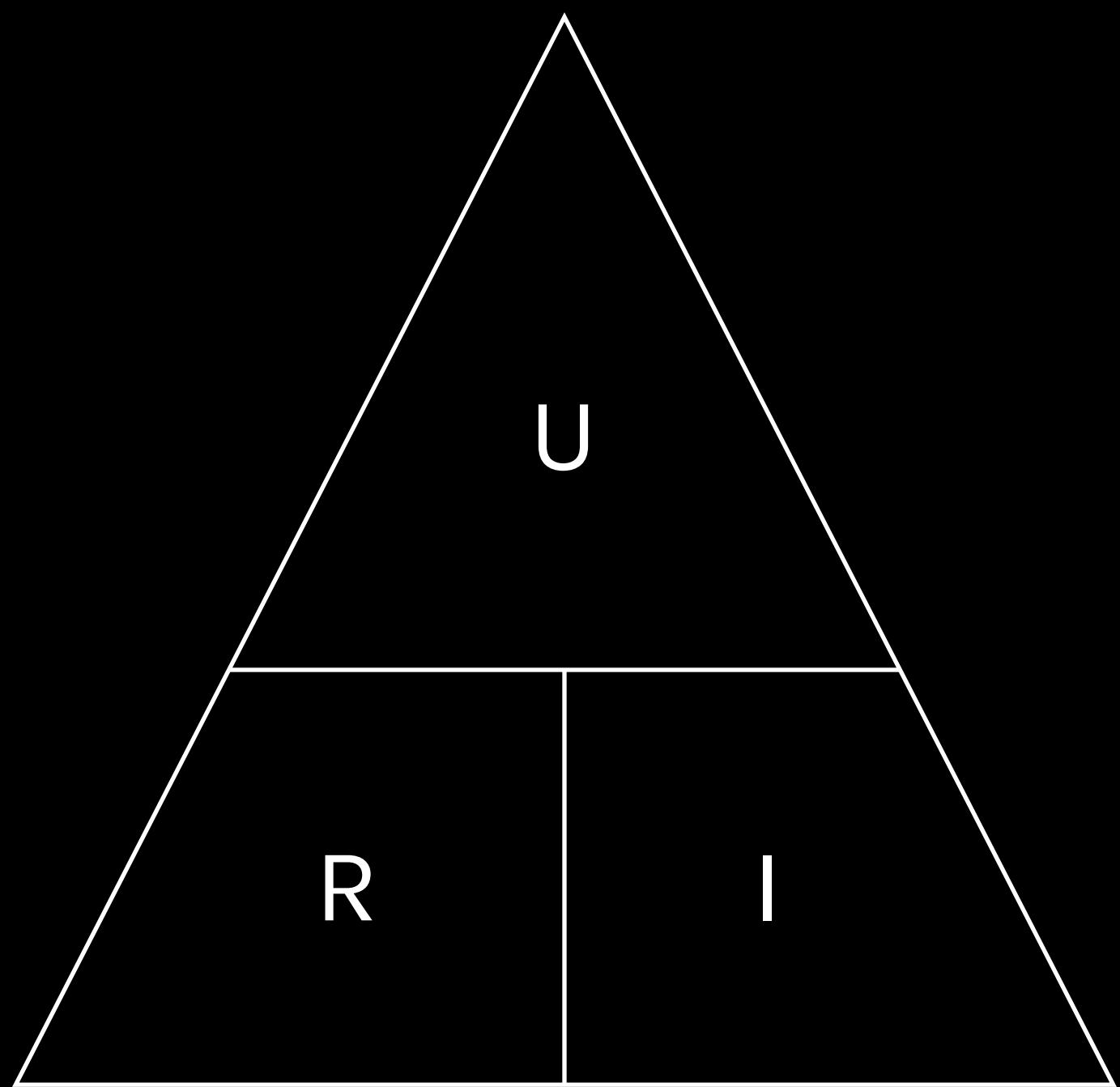
Elektrische Größen: Leistung / Watt

Watt ist die Bezeichnung oder Elektrische Einheit für Leistung. Dabei handelt es sich um den Energieumsatz pro Zeit. Demnach stellt die Einheit Watt dar, wie viel Strom entweder erzeugt oder verbraucht wird.

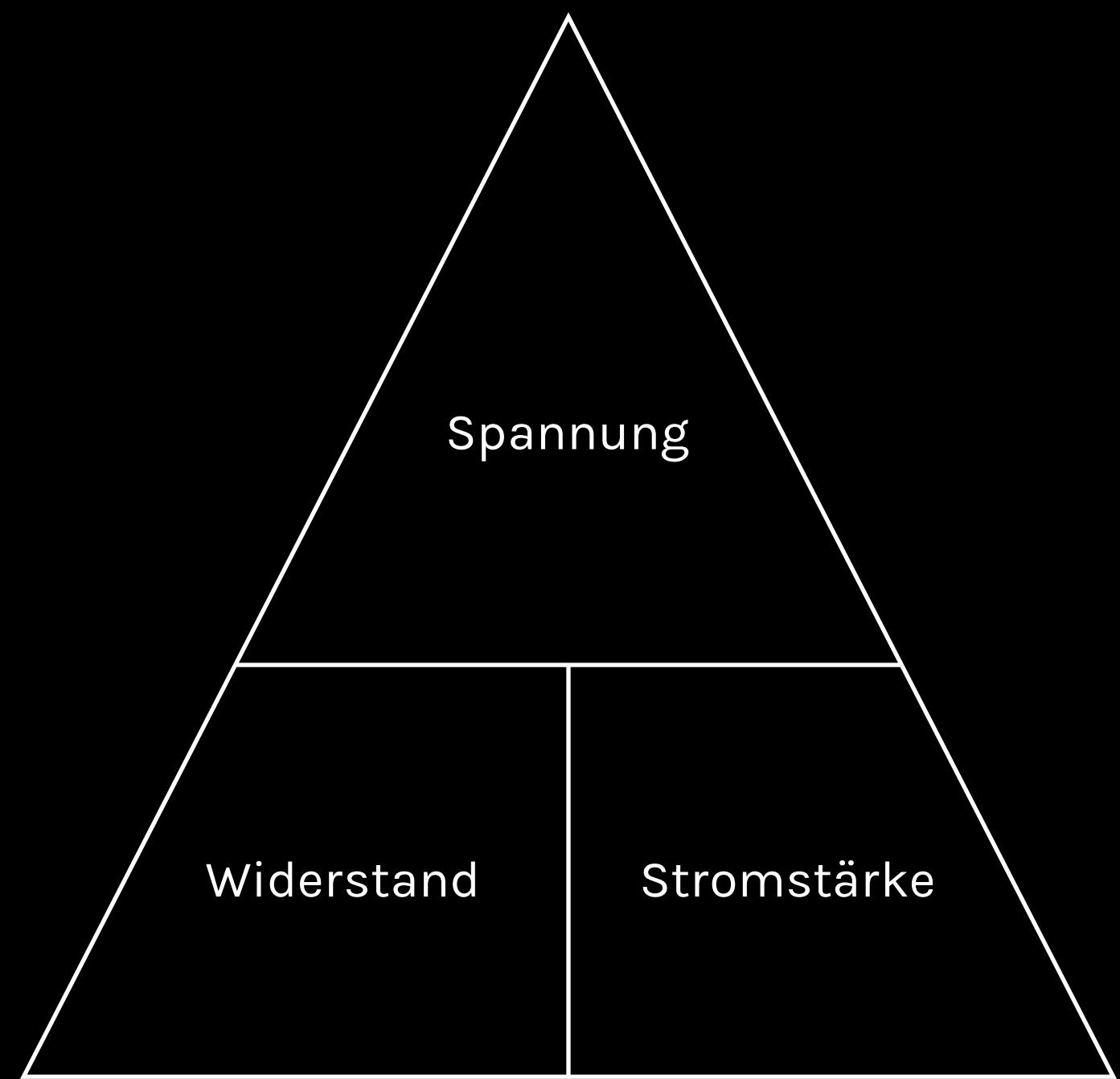
Widerstand / Ohm (R)

Der elektrische Widerstand ist der Widerstand, den Spannung in einem Stromkreis durch den elektrischen Leiter erfährt. Dadurch wird die Stromstärke reduziert. Der elektrische Widerstand funktioniert also im übertragenen Sinne wie eine mechanische Bremse.

Ohmsche Gesetz:



Ohmsche Gesetz:



Ohmsche Gesetz: $U = I \times R$

Das ohmsche Gesetz wird zur Berechnung des Verhältnisses zwischen **Spannung**, **Strom** und **Widerstand** in einem Stromkreis verwendet.

Weitere Rechnungen: Stromstärke: $I = U / R$

Widerstand: $R = U / I$

Spannung : $U = R \times I$

Arduino (Plattform): Arduino ist eine quelloffene Physical-Computing-Plattform. Sie besteht sowohl aus Soft- und Hardware. Dabei besteht die Hardware aus sogenannten I/O Boards bzw. Entwicklerboards inkl. ihren Mikrocontrollern.

Die Programmierung selbst erfolgt in einer C++ ähnlichen Programmiersprache. Der häufigste Anwendungszweck von Arduino liegt dabei in der Erstellung eigenständiger, interaktiver Objekte.

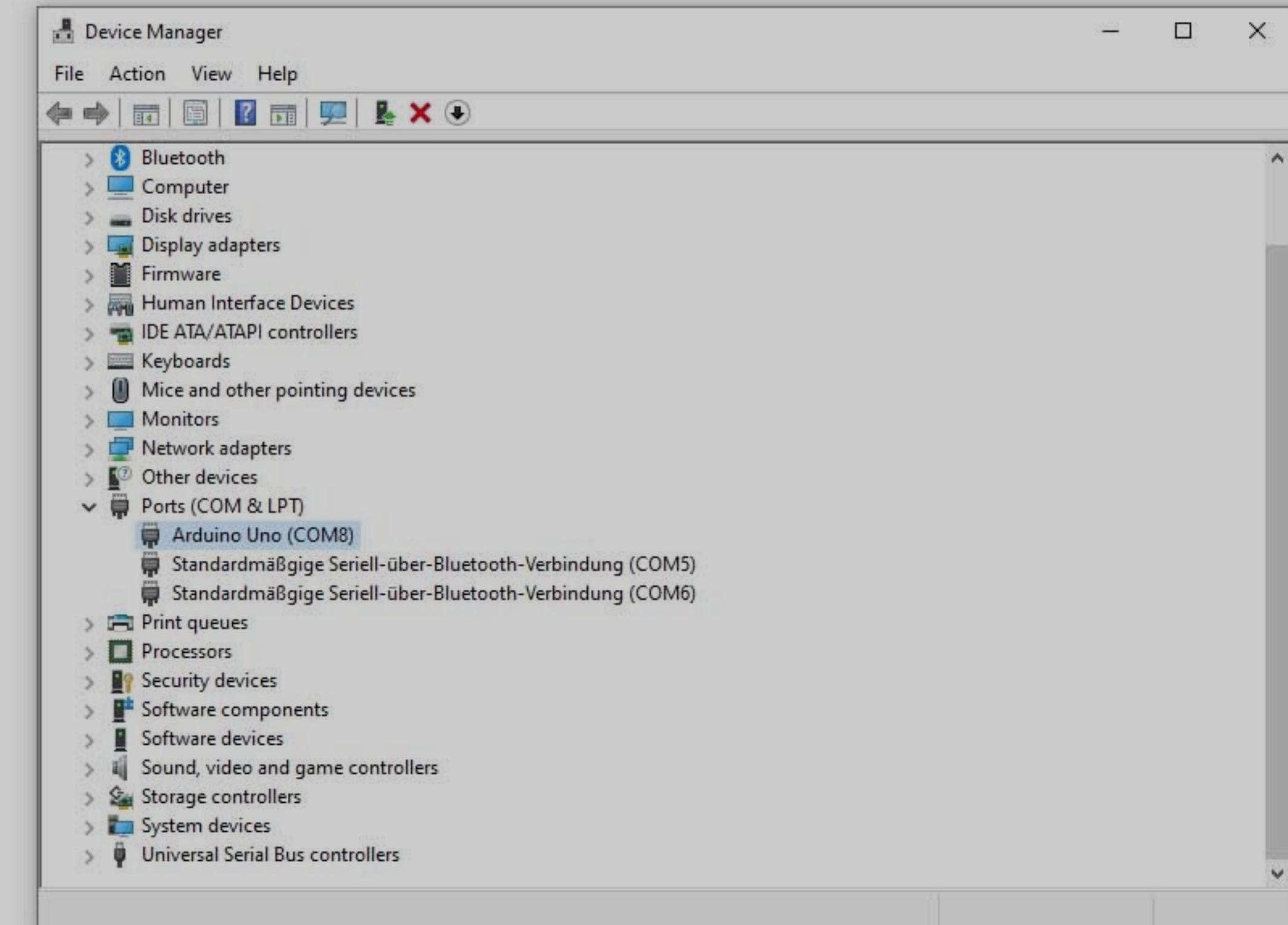
sketch_nov15a

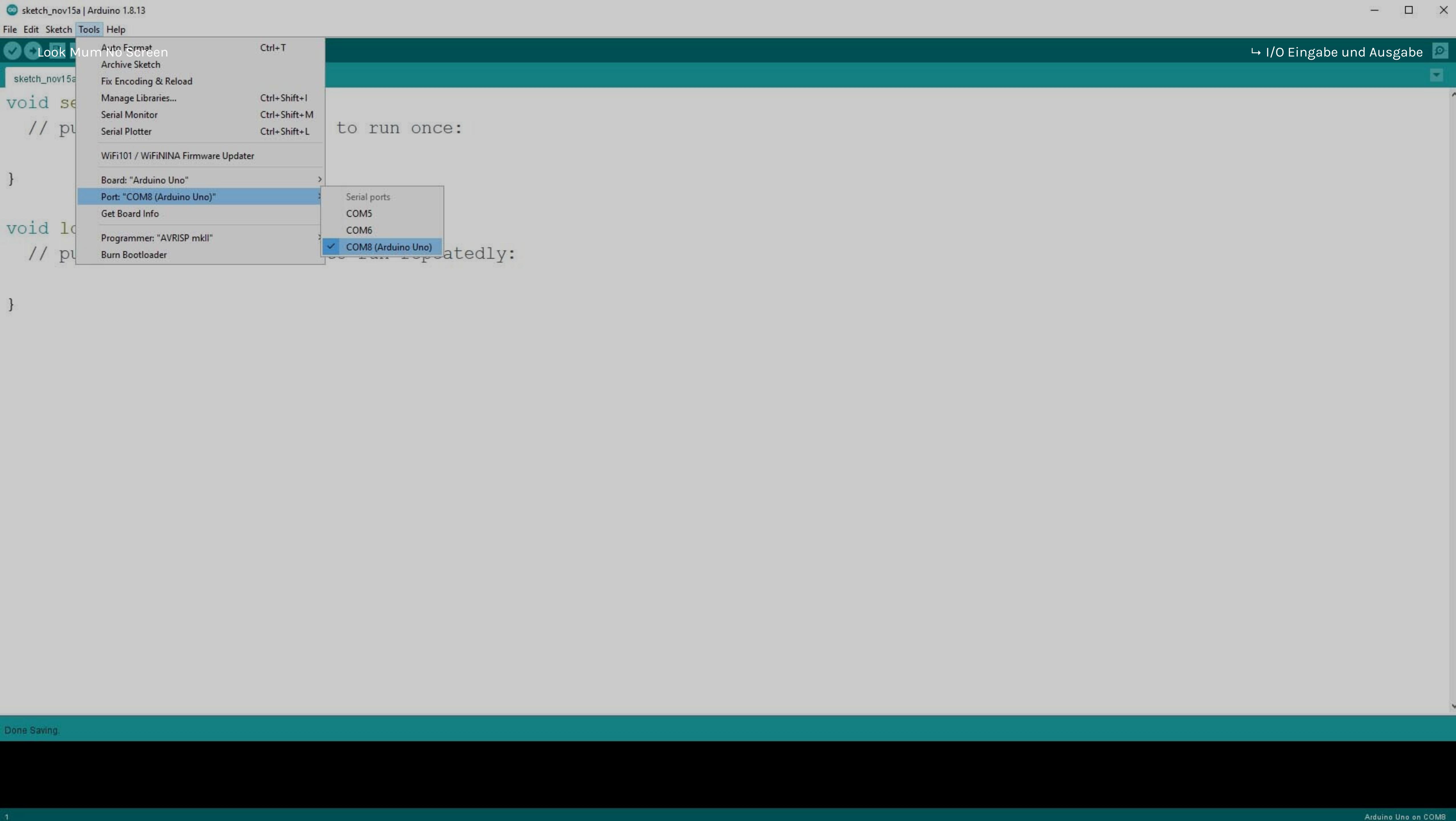
```
void setup() {  
  // put your setup code here, to run once:  
  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  
}
```

```
sketch_nov15a

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

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





sketch_nov15a

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

A screenshot of the Arduino IDE interface. The main window shows a blank sketch with the following code:

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

The sketch is titled "sketch_nov15a". Below the code editor is a serial monitor window titled "COM8". The monitor window has the following configuration:

- Autoscroll checkbox is checked.
- Show timestamp checkbox is unchecked.
- Baud rate dropdown is set to "9600 baud".
- Clear output button is visible.

A red arrow points to the close button (X) of the serial monitor window.

sketch_nov15a

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

The screenshot shows the Arduino IDE's Serial Monitor window. The title bar of the window reads "COM8". At the bottom of the window, there are three input fields: "9600 baud" (with a dropdown arrow), a text input field containing a single vertical bar character "|", and a "Send" button. To the right of the text input field is another dropdown menu labeled "No line ending".

sketch_nov15a

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

sketch_nov15a

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

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

setup() Funktion:

Die **setup()** Funktion wird einmalig aufgerufen wenn das Programm startet. Sie soll benutzt werden, um Variablen, Pin-Modi, Bibliotheken, usw. zu initialisieren.

Quelle: <https://www.arduino.cc/reference/de/language/structure/sketch/setup/>

sketch_nov15a

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

}

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

setup() Funktion:

Die **setup()** Funktion wird einmalig aufgerufen wenn das Programm startet. Sie soll benutzt werden, um Variablen, Pin-Modi, Bibliotheken, usw. zu initialisieren.

Quelle: <https://www.arduino.cc/reference/de/language/structure/sketch/setup/>

loop() Funktion:

Nach dem Erstellen einer **setup()** Funktion, die die Anfangswerte (Variablen, Pins und Bibliotheken) initialisiert, macht die Funktion **loop()** genau das, was der Name andeutet. Sie ist eine Endlosschleife, die nach jedem Durchlauf erneut aufgerufen wird. Dadurch kann das Programm Variablen verändern, Daten lesen oder darauf reagieren.

Quelle: <https://www.arduino.cc/reference/de/language/structure/sketch/loop/>

**Aufbau einer
Funktion:**

Der Aufbau einer Funktion verfolgt ein bestimmtes Schema.
Eine Funktion macht dann Sinn, wenn es sich um eine wiederholende Aufgaben bzw. Anweisung handelt.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
void loop(){  
    // put your main code here, to run repeatedly:  
  
}  
  
Typ funktionsName(parameter){  
    // schreibe deine Anweisungen hier:  
  
}
```

Aufbau einer Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.

Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5;  
    int y = x + 2;  
    return y;  
}
```

Aufbau einer Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.

Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5; ← // Variable x mit dem Wert 5  
    int y = x + 2;  
    return y;  
}
```

Aufbau einer Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.

Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5;  
    int y = x + 2; ← // Addition der Variable x + 2  
    return y;  
}
```

Aufbau einer Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.

Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5;  
    int y = x + 2;  
    return y; ← // Was wird zurück gegeben?  
}
```

Aufbau einer
Funktion:

Geschweifte Klammern definieren den Anfang und das Ende von Funktions- und Anweisungsblöcken. Diese werden ebenfalls bei **for-Schleifen** und **if-Anweisung** verwendet.

```
int calculate(){ ← // geöffnete Klammer
    int x = 5;
    int y = x + 2;
    return y;
} ← // geschlossene Klammer
```

Aufruf einer
Funktion:

Unsere Funktion wird mit dem Funktionsnamen und den beiden Klammern aufgerufen.

```
int calculate(){  
    int x = 5;  
    int y = x + 2;  
    return y;  
}  
  
calculate(); ← // Funktion wird aufgerufen
```

Variablen und Datentypen:

Durch das Erstellen einer Variable wird die Benennung eines numerischen Wertes mit einem Namen und Speicherplatz verursacht.

Der Wert der Variable kann sich kontinuierlich verändern. Im Gegensatz zu Konstanten deren Wert im Programmablauf konstant bleibt. Eine Variable muss deklariert werden und optional mit einem Wert versehen werden.

```
int x; ← // Variable x wird deklariert  
int x = 5;
```

Variablen und Datentypen:

Durch das Erstellen einer Variable wird die Benennung eines numerischen Wertes mit einem Namen und Speicherplatz verursacht.

Der Wert der Variable kann sich kontinuierlich verändern. Im Gegensatz zu Konstanten deren Wert im Programmablauf konstant bleibt. Eine Variable muss deklariert werden und optional mit einem Wert versehen werden.

```
int x;  
int x = 5; ← // Variable x wird deklariert  
              und der Wert 5 zugewiesen.
```

Variablen und Datentypen:

Durch das Erstellen einer Variable wird die Benennung eines numerischen Wertes mit einem Namen und Speicherplatz verursacht.

Der Wert der Variable kann sich kontinuierlich verändern. Im Gegensatz zu Konstanten deren Wert im Programmablauf konstant bleibt. Eine Variable muss deklariert werden und optional mit einem Wert versehen werden.

Achtung!

```
int x; // Valide Deklaration der Variable

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

int calculate(){
    x = 5; // Änderung des Wertes der Variable
    int y = x + 2;
    return y;
}
```

Variablen und Datentypen:

Durch das Erstellen einer Variable wird die Benennung eines numerischen Wertes mit einem Namen und Speicherplatz verursacht.

Der Wert der Variable kann sich kontinuierlich verändern. Im Gegensatz zu Konstanten deren Wert im Programmablauf konstant bleibt. Eine Variable muss deklariert werden und optional mit einem Wert versehen werden.

Achtung!

```
void setup(){  
    Serial.begin(9600);  
    int x; ← // Variable ausserhalb des Scopes, daher  
}                                in unserer Funktion nicht verfügbar.  
  
int calculate(){  
    x = 5; ←  
    int y = x + 2;  
    return y;  
}
```

Variablen und Datentypen:

Mit der Variable und dem Wert **int x = 5;** haben wir einen Datentypen **names Integer (int)** erstellt. Integer werden für das Speichern von Werten (ganzzahlig, 16 Bit, Werte von -32.767 bis 32.768.) ohne Dezimalkomma verwendet.

Variablen und Datentypen:

Weitere Datentypen sind:

byte

Dieser Datentyp speichert einen 8-Bit numerischen, ganzzahligen Wert ohne Dezimalkomma. Der Wert kann zwischen 0 und 255 sein.

long

Ist ein Datentyp für lange Integer mit erweiterte Größe, ohne Dezimalkomma in einem 32-Bit Wertebereich von -2,147,483,648 bis 2,147,483,647.

float

Ein Datentyp für Fließkomma Werte mit Nachkommastelle. Sie besitzen eine höhere Auflösung als Integer und werden als 32-Bit Wert in einem Bereich von -3.4028235E+38 bis 3.4028235E+38 verwendet.

Variablen und
Datentypen:

String

Dieser Datentyp speichert eine Zeichenfolge (meist Text) ab.

bool

Ein bool enthält einen von zwei Werten: **true** oder **false**. Jede bool-Variable belegt dabei einen Byte Speicher.

Achtung, es gibt noch viel mehr Datentypen! Diese Auflistung erhebt keinen Anspruch auf Vollständigkeit.

Variablen und
Datentypen -
Konstante:

#define

Unter der Zuhilfenahme der Komponente **#define**, können wir dem Programm einen konstanten Wert zuweisen, noch bevor dieses kompiliert und auf den Microkontroller geladen wird.

File Edit Sketch Tools Help

Look Mum No Screen

↳ I/O Eingabe und Ausgabe

```
sketch_nov15a §

void setup() {
    Serial.begin(9600); ←
    // put your setup code here, to run once:

}

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

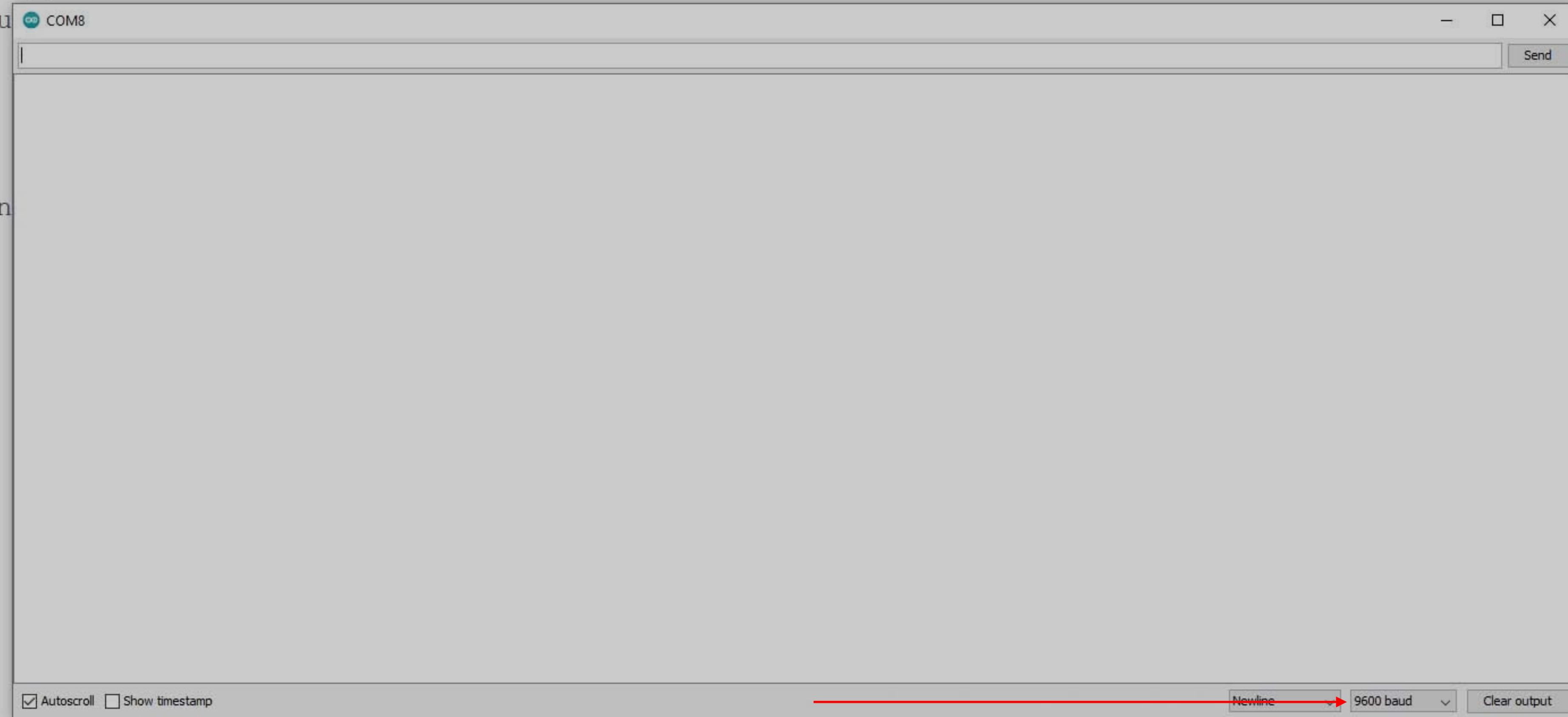
}
```

Done Saving.

`Serial.begin();` Legt die Datenrate in Bit pro Sekunde (Baud) für die serielle Datenübertragung fest.

Quelle: <https://www.arduino.cc/reference/de/language/functions/communication/serial/begin/>

```
sketch_nov15a §  
void setup() {  
    Serial.begin(9600);  
    // put your setup code here, to run  
    // before the loop() function starts  
}  
  
void loop() {  
    // put your main code here,  
    // to run repeatedly after  
    // setup() has run once  
}
```



The screenshot shows the Arduino IDE interface with a sketch named "sketch_nov15a". The sketch contains basic setup and loop functions for a serial connection at 9600 baud. Below the code editor is the Arduino Serial Monitor window, titled "COM8". The monitor has a text input field with a cursor, a "Send" button, and a scrollable output area. At the bottom of the monitor window are several configuration buttons: "Autoscroll" (checked), "Show timestamp" (unchecked), "Newline" (with a red arrow pointing to it), "9600 baud" (selected from a dropdown menu), and "Clear output".

```
sketch_nov15a §

void setup() {
    Serial.begin(9600);
    Serial.print("Hello World"); ←
    // put your setup code here, to run once:

}

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

}
```

`Serial.print();`

Druckt Daten an den seriellen Anschluss als von Menschen
lesbarer ASCII-Text. Dieser Befehl kann viele Formen annehmen.
Zahlen werden für jede Ziffer mit einem ASCII-Zeichen gedruckt.

Quelle: [https://www.arduino.cc/reference/de/language/functions/
communication/serial/print/](https://www.arduino.cc/reference/de/language/functions/communication/serial/print/)

```
sketch_nov15a
void setup() {
  Serial.begin(9600);
  Serial.print("Hello World");
  // put your setup code here, to run once:
}

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

Done compiling.

Sketch uses 1466 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 196 bytes (9%) of dynamic memory, leaving 1852 bytes for local variables. Maximum is 2048 bytes.

Look Mum No Screen

↳ I/O Eingabe und Ausgabe

sketch_nov15a

```
void setup() {  
    Serial.begin(9600) ←  
    Serial.print("Hello World");  
    // put your setup code here, to run once:  
  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
  
}
```

expected ';' before 'Serial'
exit status 1
expected ';' before 'Serial'

↳ I/O Eingabe und Ausgabe

Look Mum No Screen

```
sketch_nov15a

void setup() {
    Serial.begin(9600);
    Serial.print("Hello World");
    // put your setup code here, to run once:
}

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

Done compiling.

Sketch uses 1466 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 196 bytes (9%) of dynamic memory, leaving 1852 bytes for local variables. Maximum is 2048 bytes.

sketch_nov15a | Arduino 1.8.13

File Edit Sketch Tools Help

Look Mum No Screen ↳ I/O Eingabe und Ausgabep

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("He COM8  
    // put your setup code here, to run  
    // before the loop()  
    Hello World  
}  
  
void loop() {  
    // put your main code here  
}
```

COM8

Send

Autoscroll Show timestamp

Newline 9600 baud Clear output

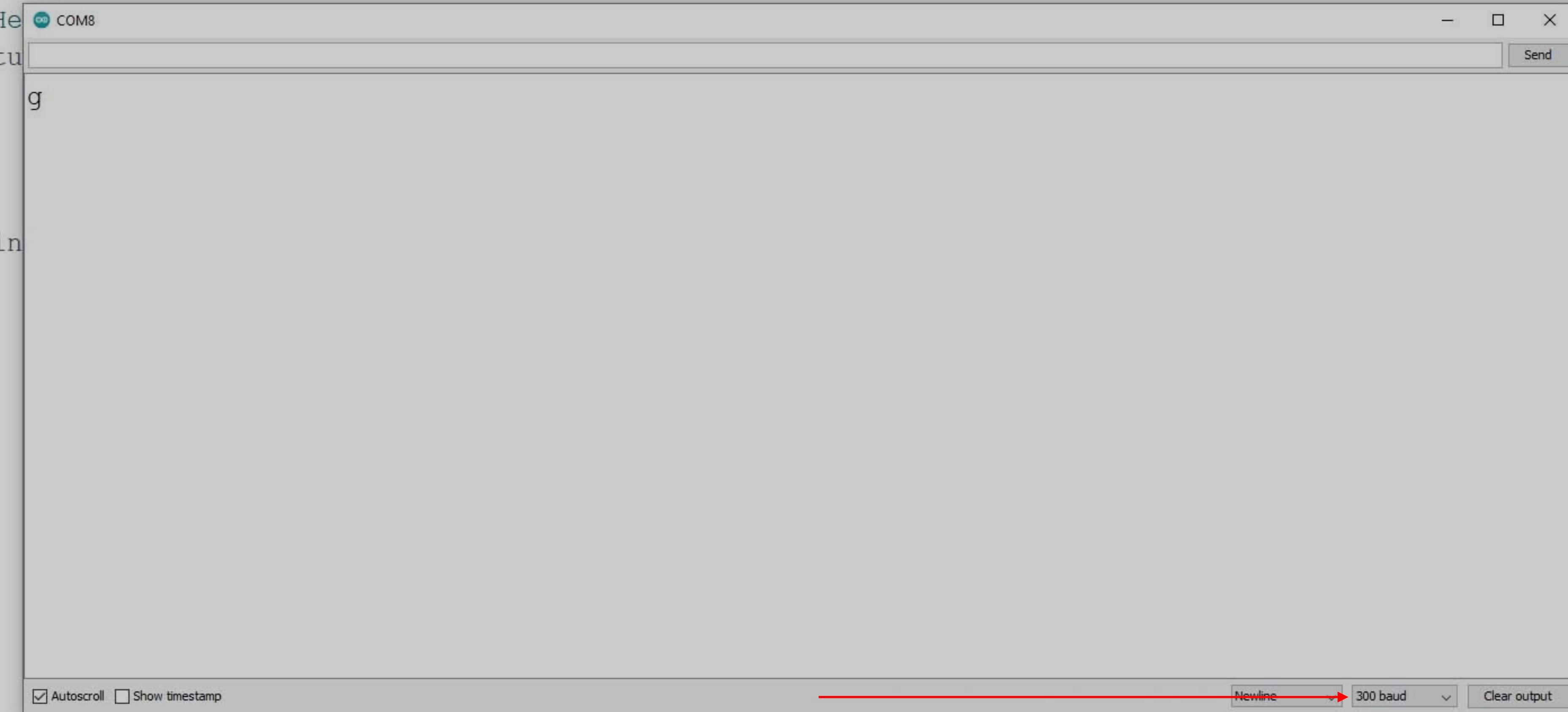
Done uploading.

Sketch uses 1466 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 196 bytes (9%) of dynamic memory, leaving 1852 bytes for local variables. Maximum is 2048 bytes.

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("He  
// put your setup  
g  
  
void loop() {  
// put your main  
}
```



COM8

Send

Autoscroll Show timestamp

Newline 300 baud Clear output

Sketch uses 1466 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 196 bytes (9%) of dynamic memory, leaving 1852 bytes for local variables. Maximum is 2048 bytes.

Look Mum No Screen

↳ I/O Eingabe und Ausgabe

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("Hello World");  
    // put your setup code here, to run once:  
  
}  
  
void loop() {  
    Serial.print("Viel zu schnell!");  
    // put your main code here, to run repeatedly:  
  
}
```

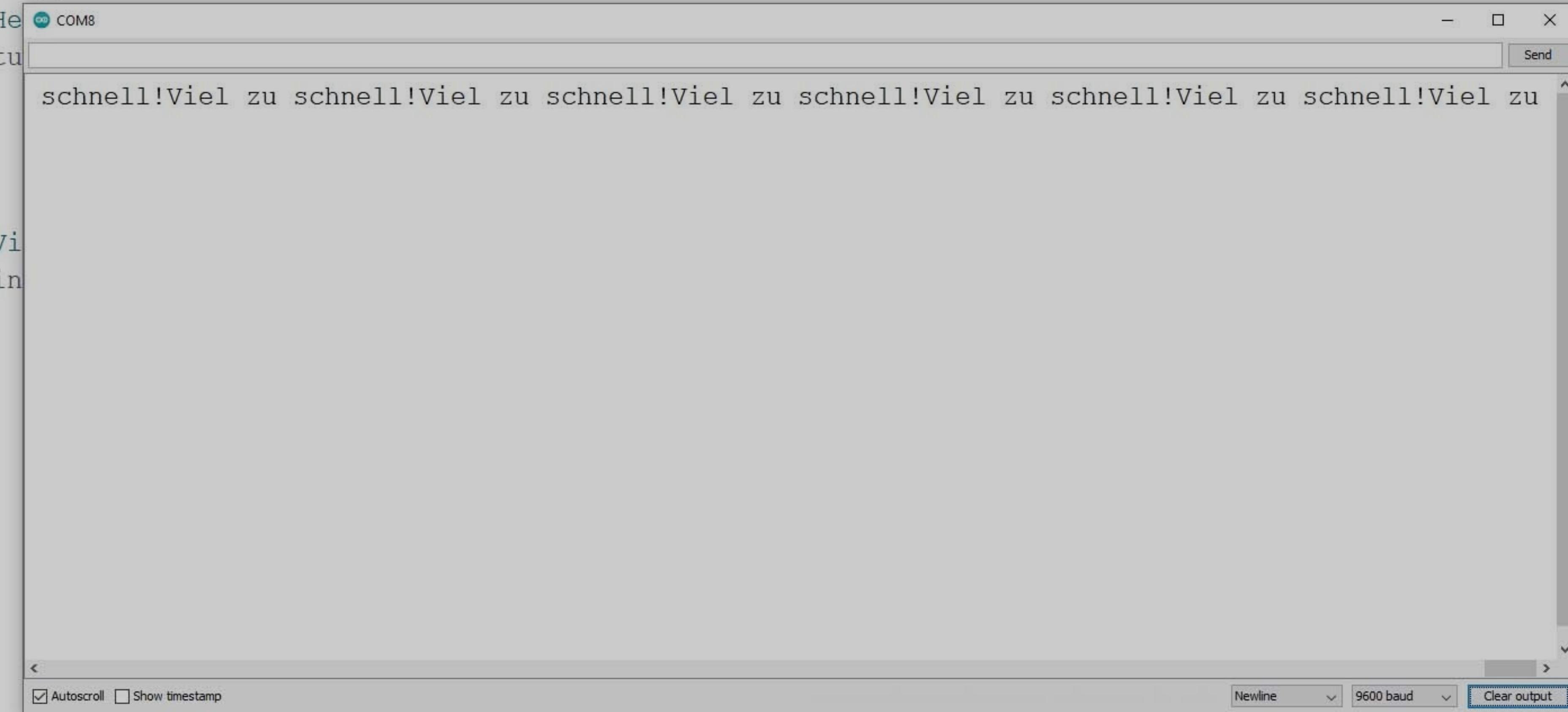
Done uploading.

Sketch uses 1500 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 214 bytes (10%) of dynamic memory, leaving 1834 bytes for local variables. Maximum is 2048 bytes.

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("He COM8  
    // put your setup code here, to run  
    schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!  
}  
  
void loop() {  
    Serial.print("Vi  
    // put your main code here  
}
```



The screenshot shows the Arduino IDE interface. In the center is a serial monitor window titled "COM8". The window displays the text "schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!Viel zu schnell!" repeatedly. Below the monitor are settings for "Autoscroll" (checked), "Show timestamp" (unchecked), "Newline" (dropdown set to "\n"), "9600 baud" (dropdown set to "9600 baud"), and a "Clear output" button. The background shows the Arduino sketch code.

Sketch uses 1500 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 214 bytes (10%) of dynamic memory, leaving 1834 bytes for local variables. Maximum is 2048 bytes.

Look Mum No Screen

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("Hello World");  
    // put your setup code here, to run once:  
  
}  
  
void loop() {  
    Serial.print("Viel zu schnell! \n");  
    // put your main code here, to run repeatedly:  
  
}
```

Sketch uses 1502 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 216 bytes (10%) of dynamic memory, leaving 1832 bytes for local variables. Maximum is 2048 bytes.

sketch_nov15a | Arduino 1.8.13

File Edit Sketch Tools Help

Look Mum No Screen ↳ I/O Eingabe und Ausgabe

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("He COM8  
    // put your setup code here, to run  
    // before the loop()  
}  
  
void loop() {  
    Serial.print("Vi COM8  
    // put your main loop code here.  
}  
  
Viel zu schnell!  
Viel zu schnell!
```

Autoscroll Show timestamp

Newline 9600 baud Clear output

Sketch uses 1502 bytes (4%) of program storage space. Maximum is 32256 bytes.

Global variables use 216 bytes (10%) of dynamic memory, leaving 1832 bytes for local variables. Maximum is 2048 bytes.

```
sketch_nov15a

void setup() {
    Serial.begin(9600);
    Serial.print("Hello World \n");
    // put your setup code here, to run once:

}

void loop() {
    delay(1000); ←
    Serial.print("Das ist gut! \n");
    // put your main code here, to run repeatedly:

}
```

Done Saving.

Sketch uses 1646 bytes (5%) of program storage space. Maximum is 32256 bytes.

Global variables use 212 bytes (10%) of dynamic memory, leaving 1836 bytes for local variables. Maximum is 2048 bytes.

```
delay();
```

Unterbricht das Programm für die als Parameter angegebene Zeit (in Millisekunden).

Quelle: <https://www.arduino.cc/reference/de/language/functions/time/delay/>

sketch_nov15a | Arduino 1.8.13

File Edit Sketch Tools Help

Look Mum No Screen ↳ I/O Eingabe und Ausgabe

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("He COM8  
    // put your setup code here, to run  
    // before the loop()  
}  
  
void loop() {  
    delay(1000);  
    Serial.print("Da  
    // put your main code here, to run  
    // after setup()  
}  
  
Das ist gut!  
Das ist gut!
```

Autoscroll Show timestamp

Newline 9600 baud Clear output

Sketch uses 1648 bytes (5%) of program storage space. Maximum is 32256 bytes.

Global variables use 214 bytes (10%) of dynamic memory, leaving 1834 bytes for local variables. Maximum is 2048 bytes.

Unsere Funktion
testen:

```
int calculate(){  
    int x = 5;  
    int y = x + 2;  
    return y;  
}
```

```
sketch_nov15a

void setup() {
    Serial.begin(9600);
    // put your setup code here, to run once:

}

void loop() {
    delay(1000);
    Serial.println(calculate());
    // put your main code here, to run repeatedly:

}

int calculate(){
    int x = 5;
    int y = x + 2;
    return y;
}
```

Done Saving.

```
sketch_nov15a

void setup() {
    Serial.begin(9600);
    // put your setup code here, to run once:

}

void loop() {
    delay(1000);
    Serial.println(calculat());e
    // put your main code here, to run repeatedly:

}

int calculate(){
    int x = 5;
    int y = x + 2;
    return y;
}
```

```
sketch_nov15a

void setup() {
    Serial.begin(9600);
    // put your setup code here

}

void loop() {
    delay(1000);
    Serial.println("Hello World!");
    // put your main loop code here

}

int calculate() {
    int x = 5;
    int y = x + 2;
    return y;
}
```

Include library:

Die Arduino-Umgebung kann wie die meisten Programmierplattformen durch den Einsatz von Bibliotheken erweitert werden. Bibliotheken bieten zusätzliche Funktionen.

Quelle: <https://www.arduino.cc/reference/en/libraries/>

sketch_nov15a | Arduino 1.8.13

File Edit Sketch Tools Help

Look Mum No Screen Auto Format Ctrl+T

Archive Sketch

Fix Encoding & Reload

Manage Libraries... Ctrl+Shift+I

Serial Monitor Ctrl+Shift+M

Serial Plotter Ctrl+Shift+L

WiFi101 / WiFiNINA Firmware Updater

Board: "Arduino Leonardo" >

Port: "COM11 (Arduino Leonardo)" >

Get Board Info

Programmer: "AVRISP mkII" >

Burn Bootloader

void setup() {
 Serial.begin(9600);
 // put your setup code here, to run once:
}

void loop() {
 delay(1000);
 Serial.println(calculate());
 // put your main code here, to run repeatedly:
}

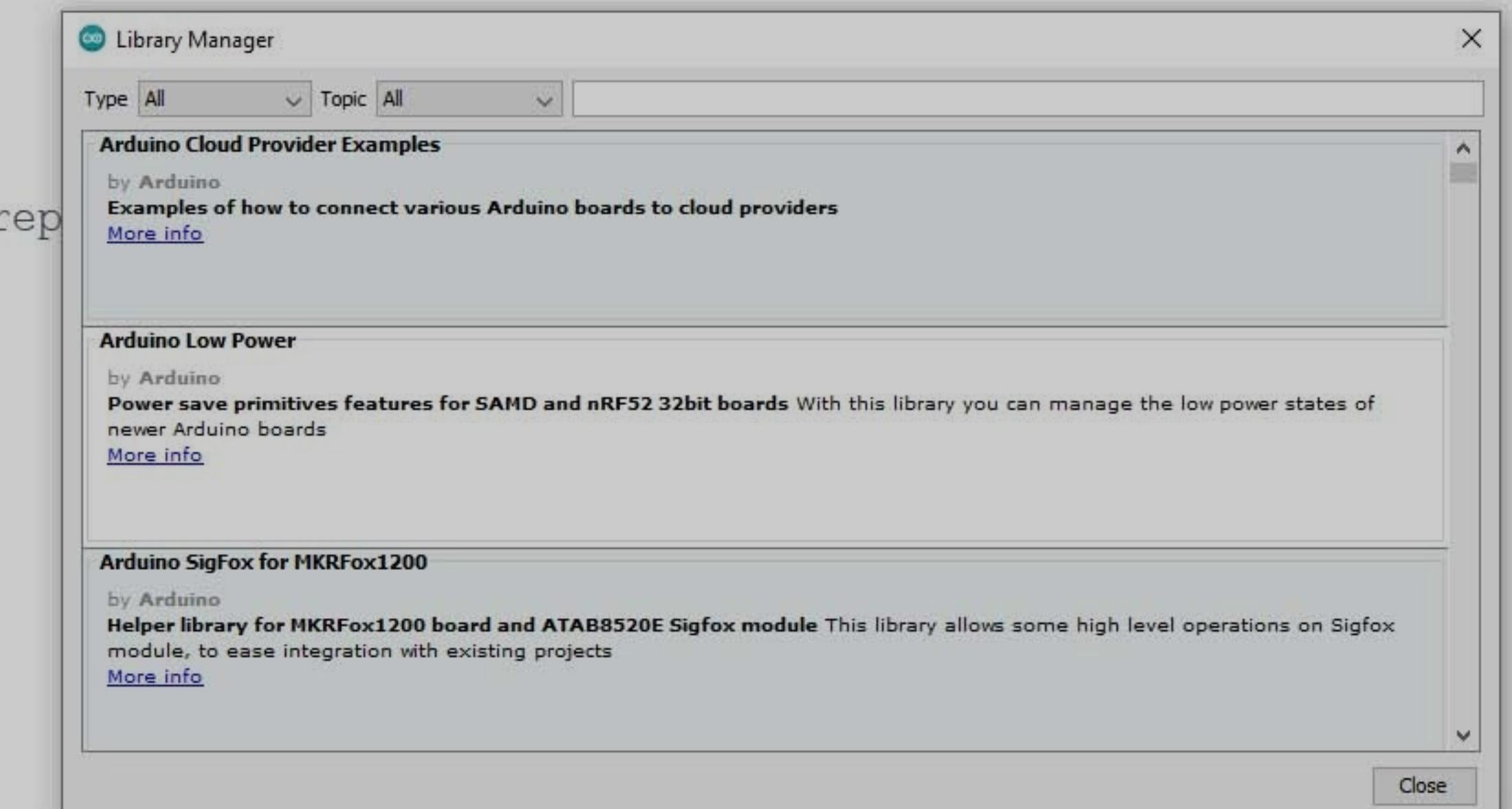
int calculate() {
 int x = 5;
 int y = x + 2;
 return y;
}

I/O Eingabe und Ausgabe

Arduino Leonardo on COM11

sketch_nov15a

```
void setup() {  
    Serial.begin(9600);  
    // put your setup code here, to run once:  
  
}  
  
void loop() {  
    delay(1000);  
    Serial.println(calculate());  
    // put your main code here, to run rep  
  
}  
  
int calculate(){  
    int x = 5;  
    int y = x + 2;  
    return y;  
}
```



sketch_nov15a | Arduino 1.8.13

File Edit Sketch Tools Help

✓ + Look Mum No Screen Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Sketch void Show Sketch Folder Ctrl+K
Serial Include Library Add File...
once:
void loop() {
 delay(1000);
 Serial.println(calculat...
 // put your main cod...
}

int calculate(){
 int x = 5;
 int y = x + 2;
 return y;
}

Manage Libraries... Ctrl+Shift+L
Add .ZIP Library...
Arduino libraries
Arduino_JSON
Bridge
EEPROM
Esplora
Ethernet
Firmata
GSM
HID
Keyboard
LiquidCrystal
Mouse
Robot Control
Robot IR Remote
Robot Motor
SD
SPI
Servo
SoftwareSerial
SpacebrewYun
Stepper
TFT
Temboo
WiFi
Wire
Contributed libraries
DoxygenExample
EasySMU
LT3965
LTC1592
LTC1859
LTC1867
LTC2302
LTC2305
LTC2308
LTC2309
LTC2315

↪ I/O Eingabe und Ausgabe

Arduino Leonardo on COM11

sketch_nov15a §

```
#include <Arduino_JSON.h>

void setup() {
    Serial.begin(9600);
    // put your setup code here, to run once:

}

void loop() {
    delay(1000);
    Serial.println(calculate());
    // put your main code here, to run repeatedly:

}

int calculate() {
    int x = 5;
    int y = x + 2;
    return y;
}
```

If-Abfragen und ihre Operatoren:

Die If-Abfrage testet, ob eine Bedingung wahr oder nicht wahr ist. Wenn eine Abfrage wahr ist, wird alles was sich zwischen den geschweiften Klammern steht, ausgeführt. Wenn unwahr, also **else**, dann wird das selbe Prinzip verfolgt.

```
int y = 7;  
  
if ( y > 8 ){  
    Serial.println("Die Zahl ist größer als 8");  
}else{  
    Serial.println("Die Zahl ist kleiner als 8");  
}
```

If-Abfragen und ihre Operatoren:

In der gerade gezeigten If-Abfrage schauen wir, ob unsere Variable mit dem Wert 7, größer als die Zahl 8 ist. Natürlich gibt es noch weitere Vergleichsmöglichkeiten um herauszufinden, ob eine Bedingung zutrifft.

- !=** (ungleich)
- <** (kleiner)
- <=** (kleiner oder gleich)
- ==** (gleich)
- >** (größer)
- >=** (größer oder gleich)

If-Abfragen und ihre Operatoren:

Wenn aber mehr als eine Bedingung zutreffen muss, um zu prüfen, ob unsere “Gesamtbedingung” wahr ist, dann spricht man von einer UND-Verknüpfung.

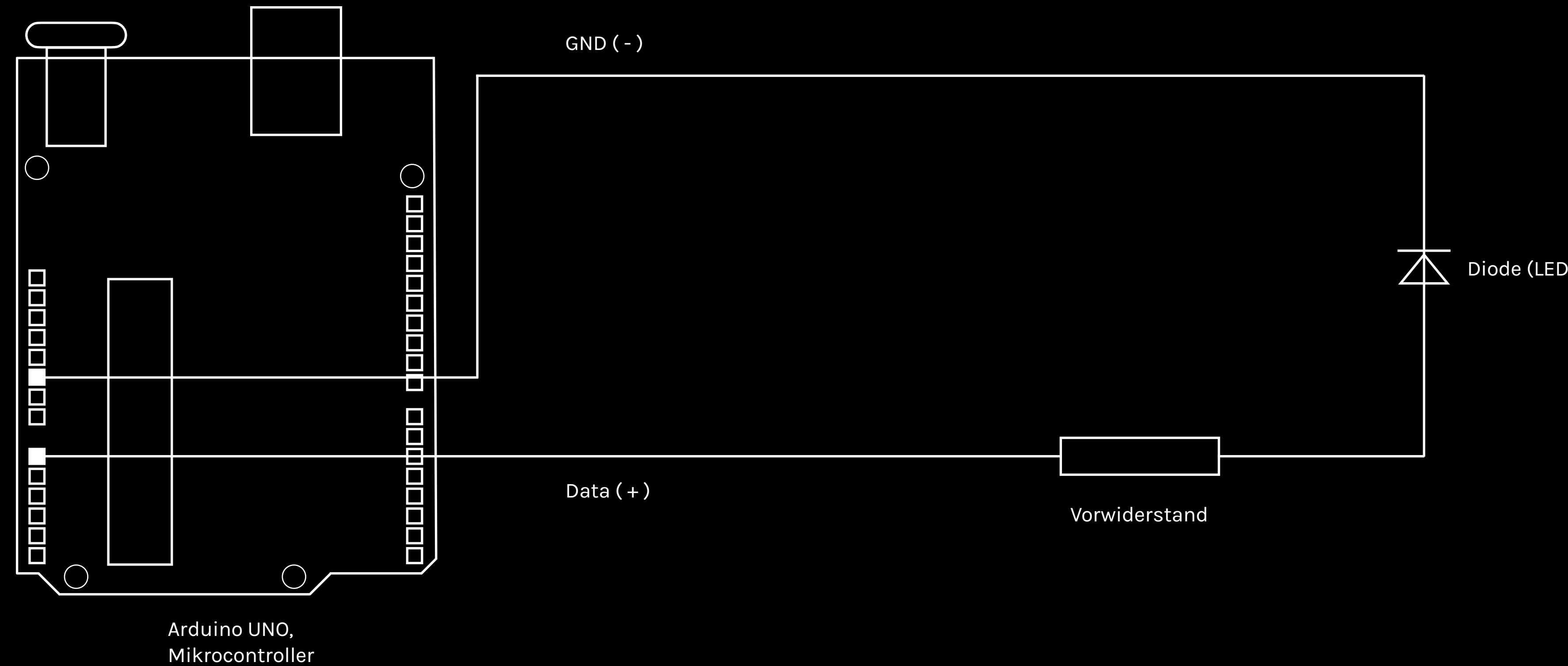
```
if ( 5<=value && value<=10 ){  
    Serial.println("Hello World");  
}
```

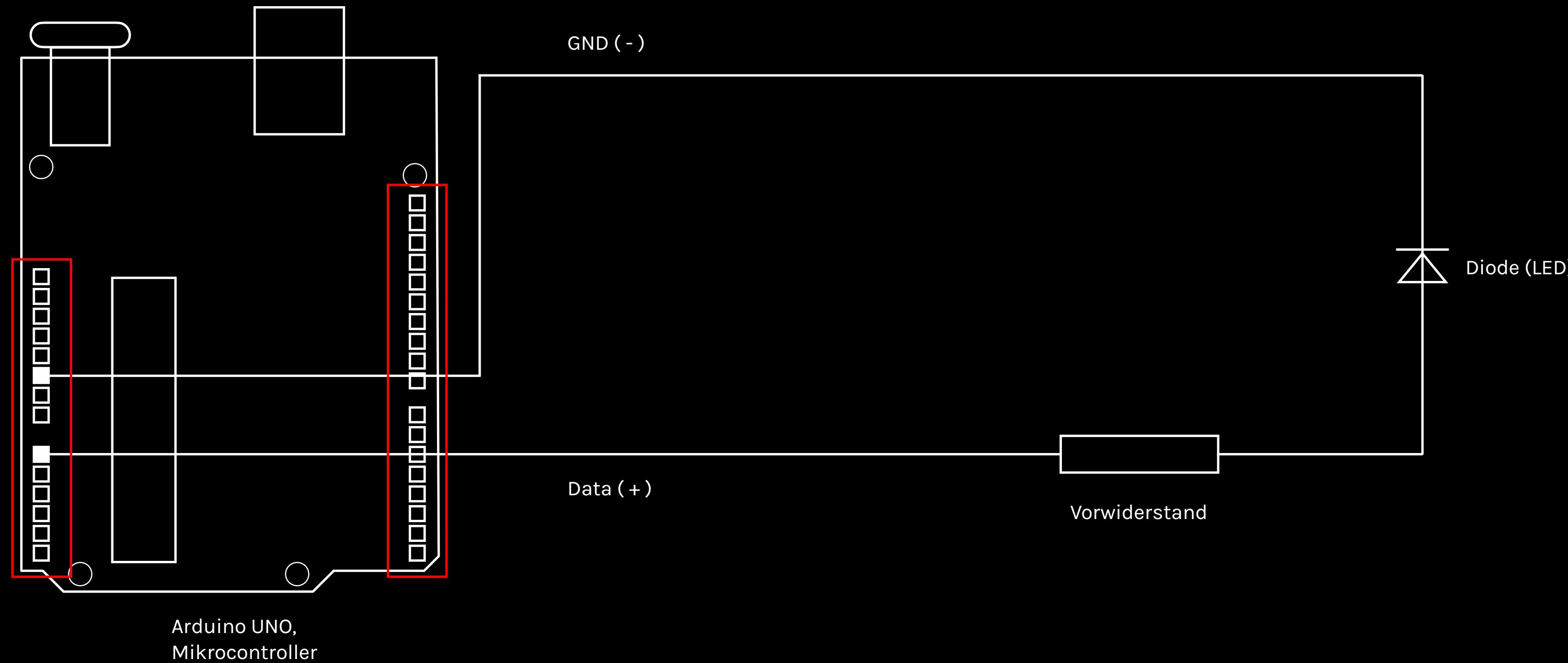
If-Abfragen und ihre Operatoren:

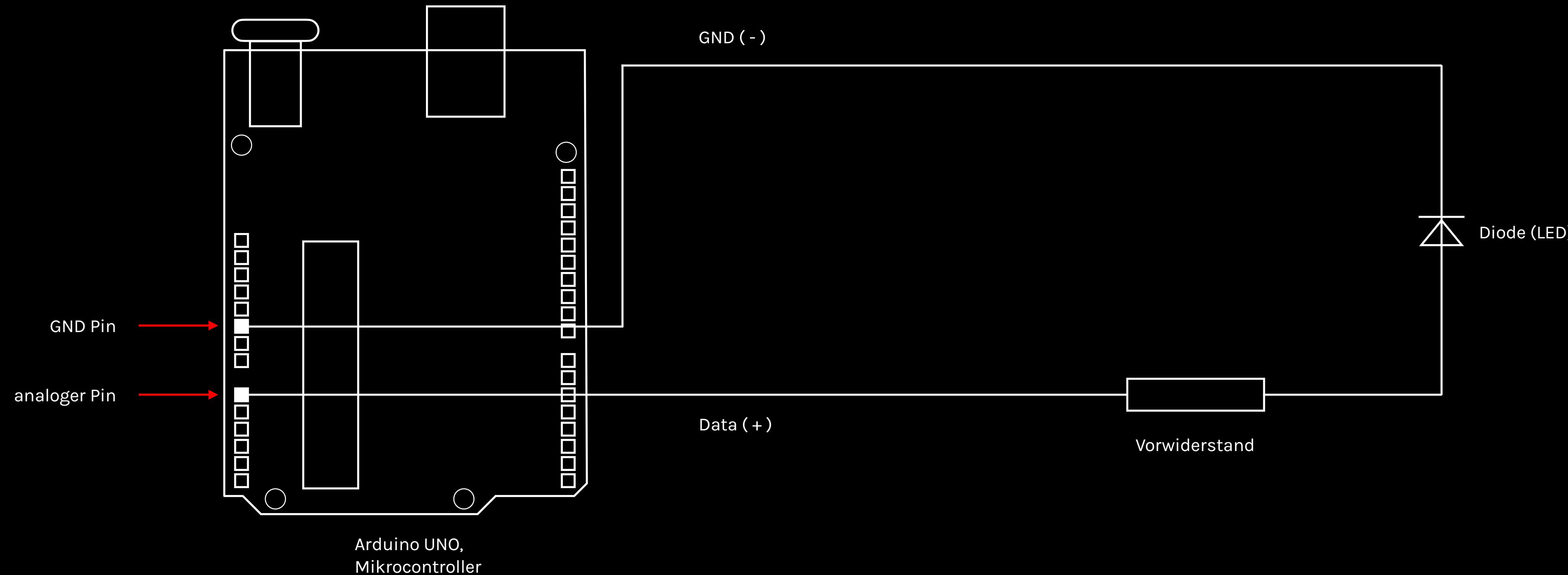
Um zu prüfen, ob eine der beiden Bedingungen zutrifft, bzw. eine Bedienung zum Teil wahr ist, müssen wir auf eine ODER-Verknüpfung zurückgreifen.

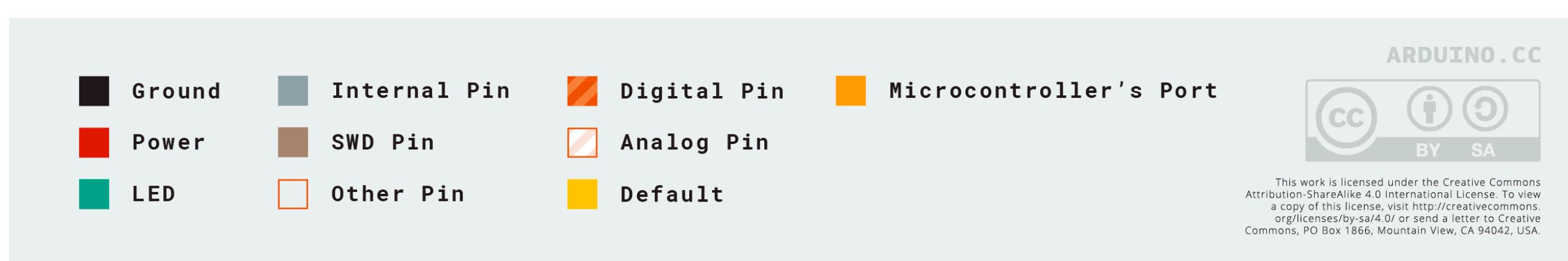
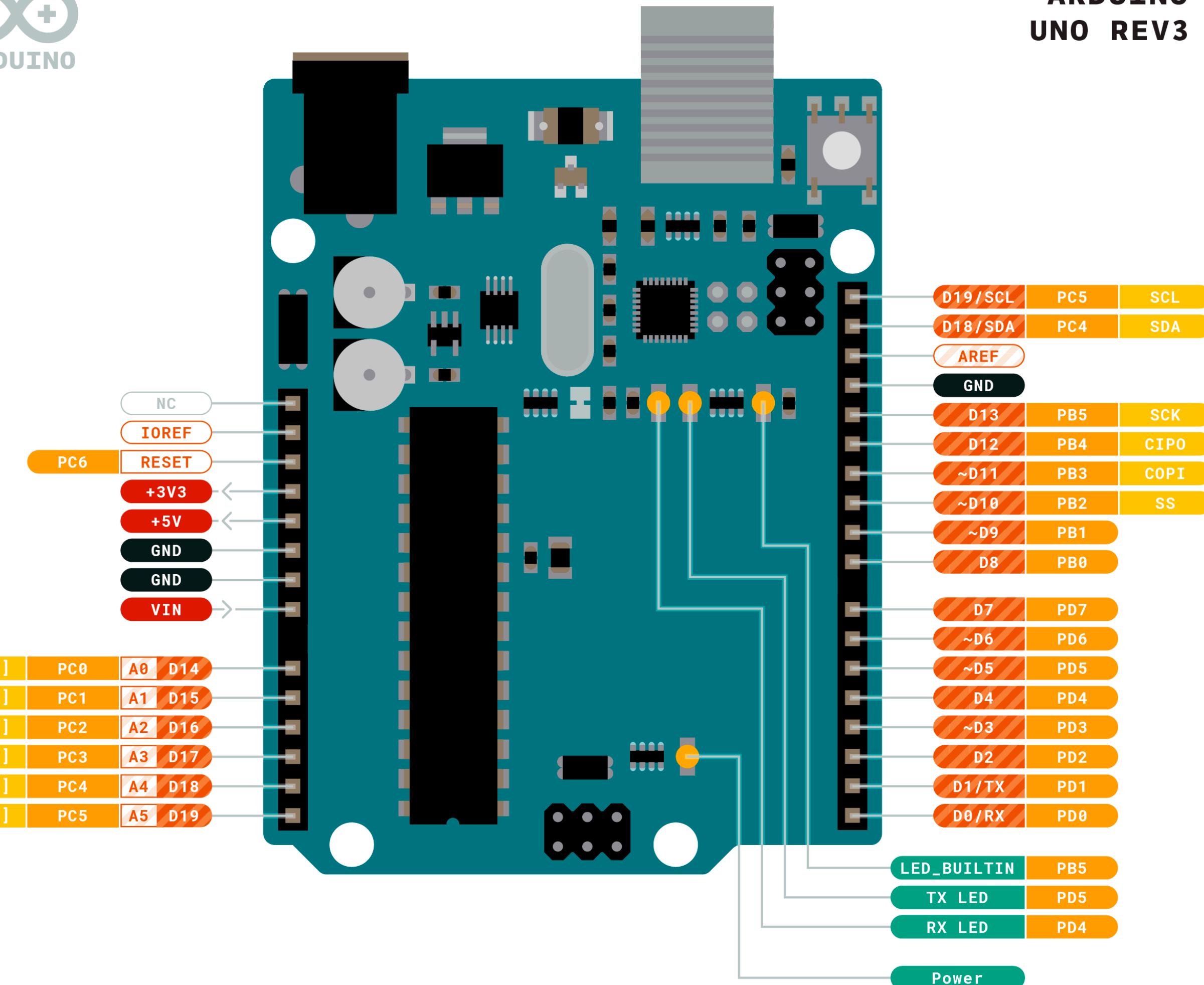
```
if ( 5>value || value>10 ){
    Serial.println("Hello World");
}
```

Achtung, es gibt noch weitere Vergleichsoperatoren sowie boolsche Verknüpfungen! Diese Auflistung erhebt keinen Anspruch auf Vollständigkeit.








**ARDUINO
UNO REV3**


Pin-Mode:

Innerhalb der `setup()` Funktion können den wir den Pin-Mode konfigurieren. Dabei wird zwischen INPUT und OUTPUT unterschieden. In unserem gezeigten Beispiel ist unser analog Pin (A0) ein OUTPUT.

```
#define led_pin A0 // Konstante led_pin zeigt auf Pin A0

void setup(){
    Serial.begin(9600);
    pinMode(led_pin, OUTPUT);
    // put your setup code here, to run once:

}
```

Analog-Write:

In der loop() Funktion nutzen wir analogWrite() und übergeben die Variable led_pin und schreiben den Wert **255**. In der zweiten Zeile schreiben wir den Wert **100**, die LED scheint gedimmt.

```
#define led_pin A0 // Konstante led_pin zeigt auf Pin A0

void setup(){
    Serial.begin(9600);
    pinMode(led_pin, OUTPUT);
    // put your setup code here, to run once:

}

void loop(){
    → analogWrite(led_pin, 255); // Wir schreiben den Wert 255, LED leuchtet
    → analogWrite(led_pin, 100); // Wir schreiben den Wert 100, LED ist "gedimmt"
}
```

Digital-Write:

In der loop() Funktion nutzen wir digitalWrite() und übergeben die Variable led_pin und schreiben **HIGH**. In der zweiten Zeile schreiben wir **LOW**, die LED ist aus.

```
#define led_pin D7 // Konstante led_pin zeigt auf Pin D7
```

```
void setup(){
    Serial.begin(9600);
    pinMode(led_pin, OUTPUT);
    // put your setup code here, to run once:

}

void loop(){
    → digitalWrite(led_pin, HIGH); // Wir setzen HIGH, LED ist an
    → digitalWrite(led_pin, LOW); // Wir setzen LOW, LED ist aus
}
```

Digital-Read:

Unter der Verwendung von `digitalRead()` können wir den Wert / Zustand eines Pins abfragen. Wir definieren eine Konstante auf einem digitalen Pin (digitaler Pin, entweder **HIGH** oder **LOW** / **0** oder **1**) und fragen in der `loop()` Funktion ab.

```
#define button_pin D8 // Konstante button_pin zeigt auf Pin D8

void setup(){
    Serial.begin(9600);
    pinMode(button_pin, OUTPUT);
    // put your setup code here, to run once:

}

void loop(){
    → int button_value = digitalRead(button_pin);

    → if( button_pin == 1){
        → Serial.println("Button wurde gedrückt");
    }
}
```

Links:

<https://create.arduino.cc/>
https://www.youtube.com/watch?v=CqrQmQqpHXc&ab_channel=Make%3A
<https://www.elektronik-kompendium.de/>
<https://www.mikrocontroller.net/>
<https://www.mikrocontroller.net/topic/261692>

[http://www.netzmafia.de/skripten/hardware/Arduino/
Arduino_Programmierhandbuch.pdf](http://www.netzmafia.de/skripten/hardware/Arduino/Arduino_Programmierhandbuch.pdf)

Material:

<https://github.com/tmjns/Look-Mum-No-Screen>