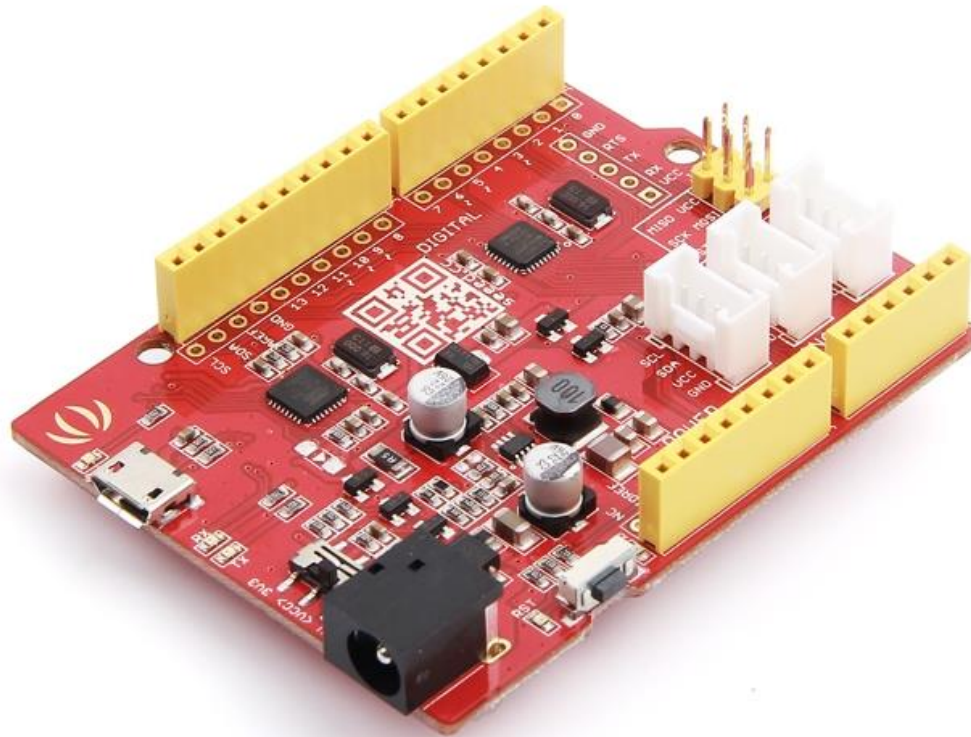


# Smarte Halloween Lampen

smile  
— IT —  
SMART · FUTURE · ME

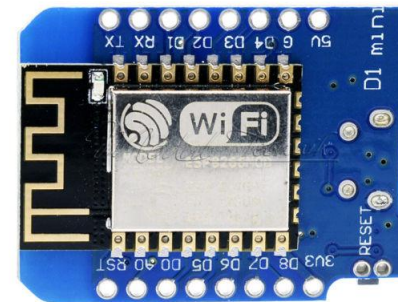


# Hands-On Arduino

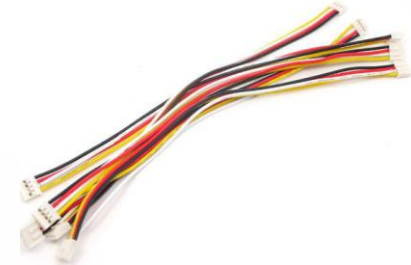


Der Arduino ist ein Mini-Computer

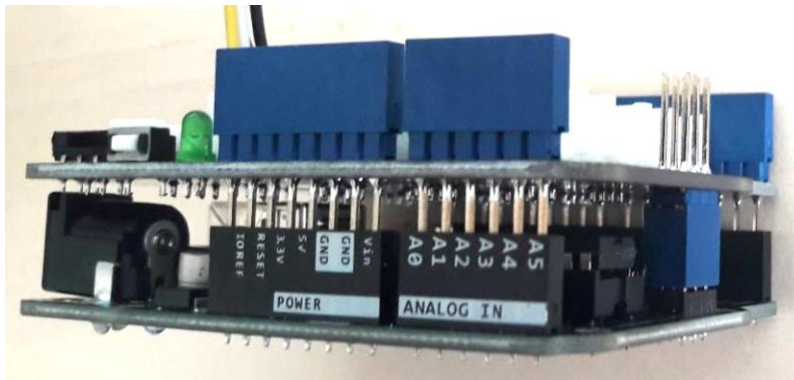
- Mit dem Arduino lassen sich schnell und einfach Sensoren und LEDs oder Motoren ansteuern.
- Durch den Arduino vereinfacht sich der Umgang mit Elektronik



# Hands-On Arduino



# Hands-On Arduino

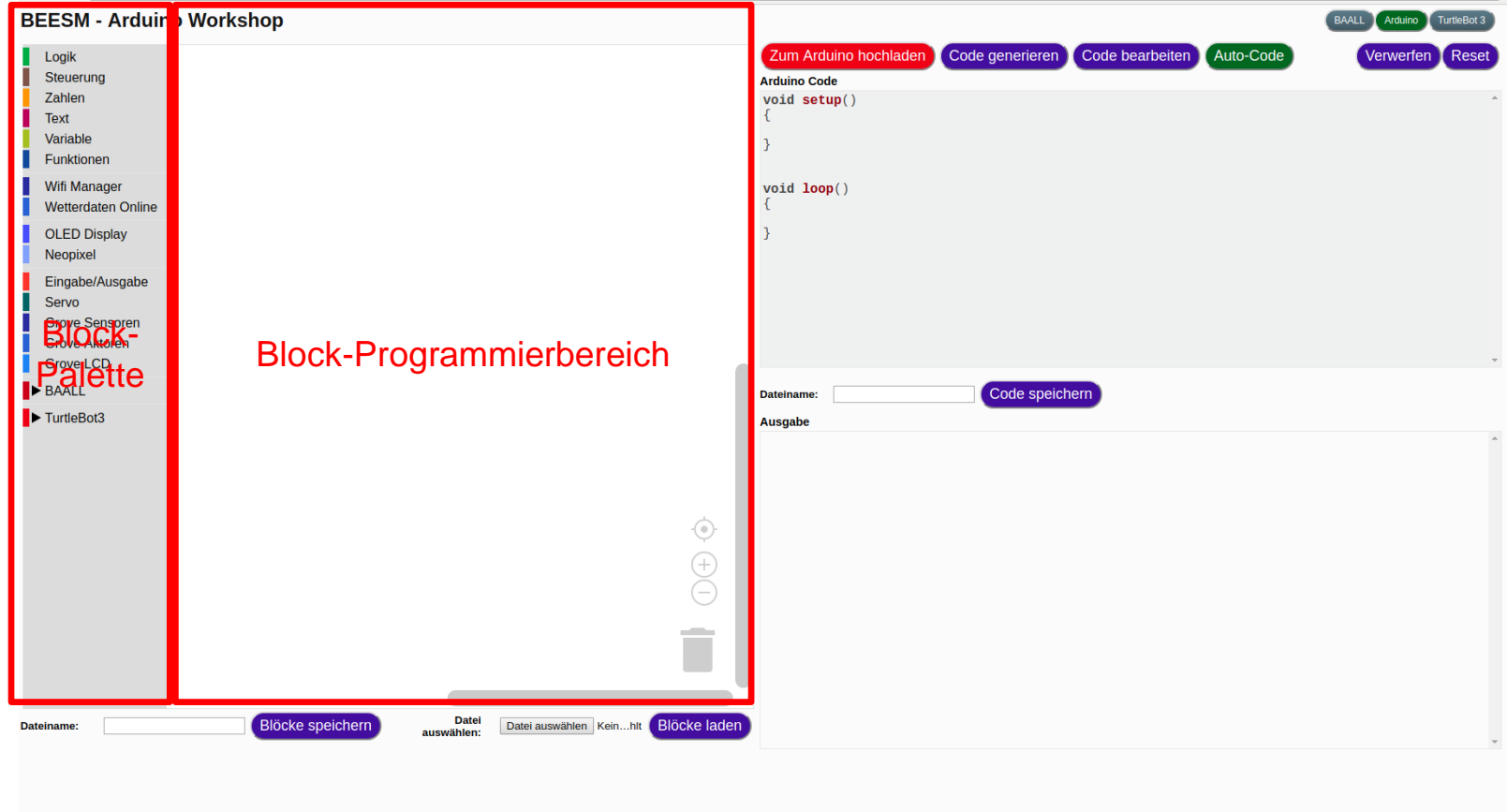


## Steckboard

- Steckt das Steckboard auf den Arduino auf, alle „Stifte“ müssen in die entsprechende Gegenseite
- Der kleine Schalter links unten muss auf 5V (rechte Schalterstellung) stehen

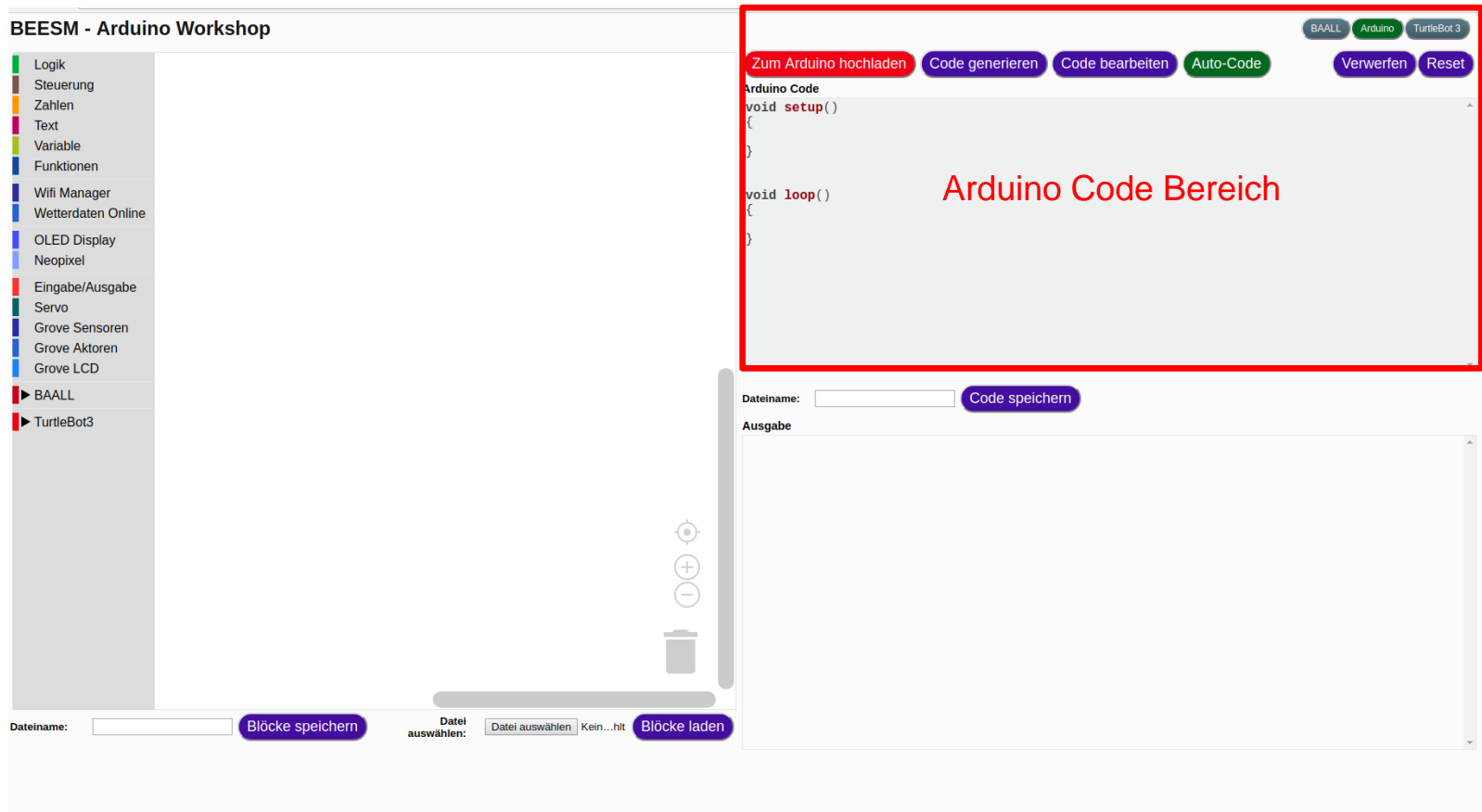


# Die Entwicklungsumgebung: BEEESM



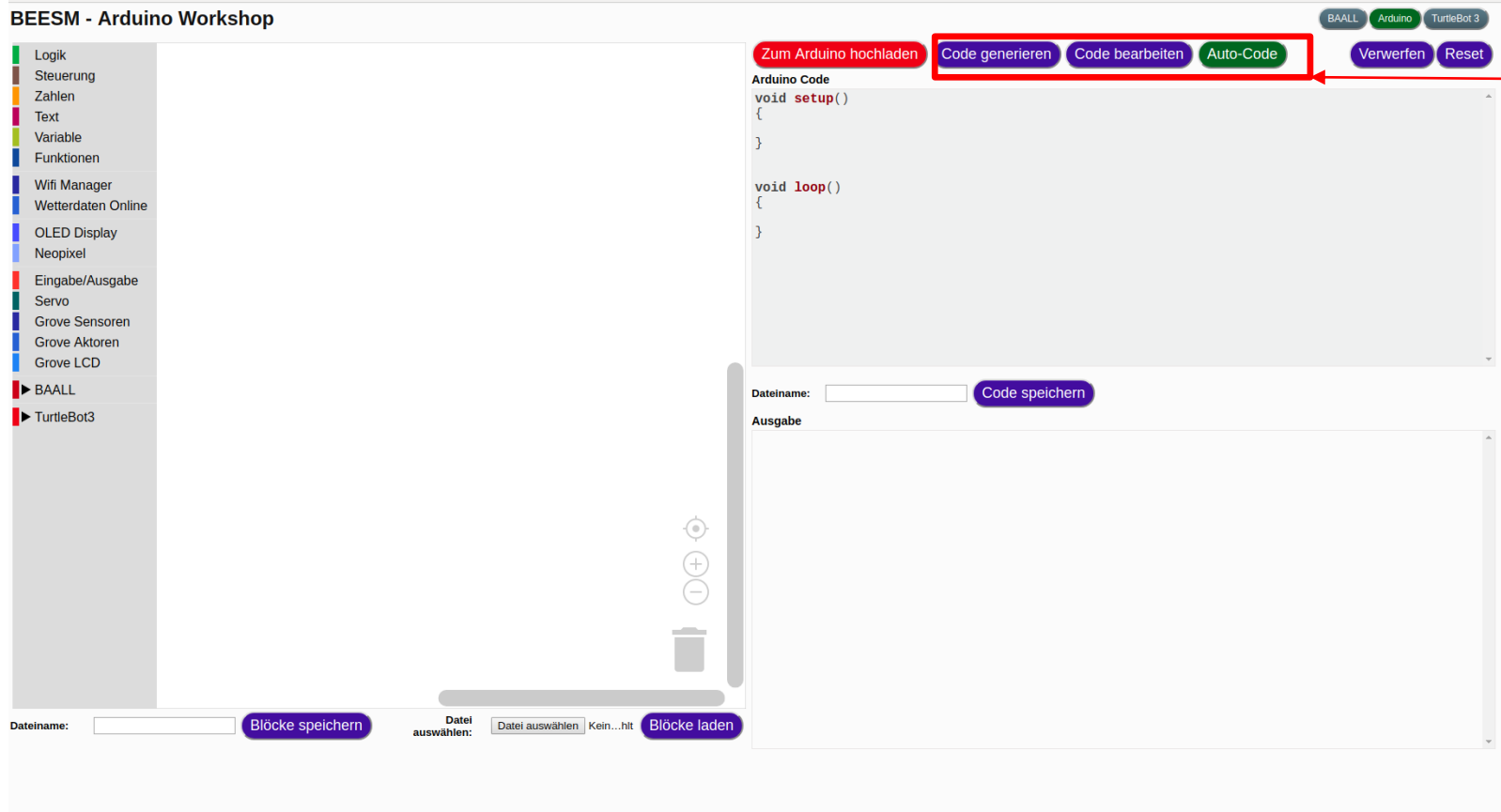
In der **Blockpalette**  
und im  
**Programmierbereich**  
arbeiten wir und  
stellen unser  
Programm aus  
verschiedenen  
Blöcken zusammen.

# Die Entwicklungsumgebung: BEEISM



Im **Code Bereich** sehen wir den Code, der aus diesen Blöcken, die wir im Block-Programmierbereich zusammengestellt haben, generiert wird.

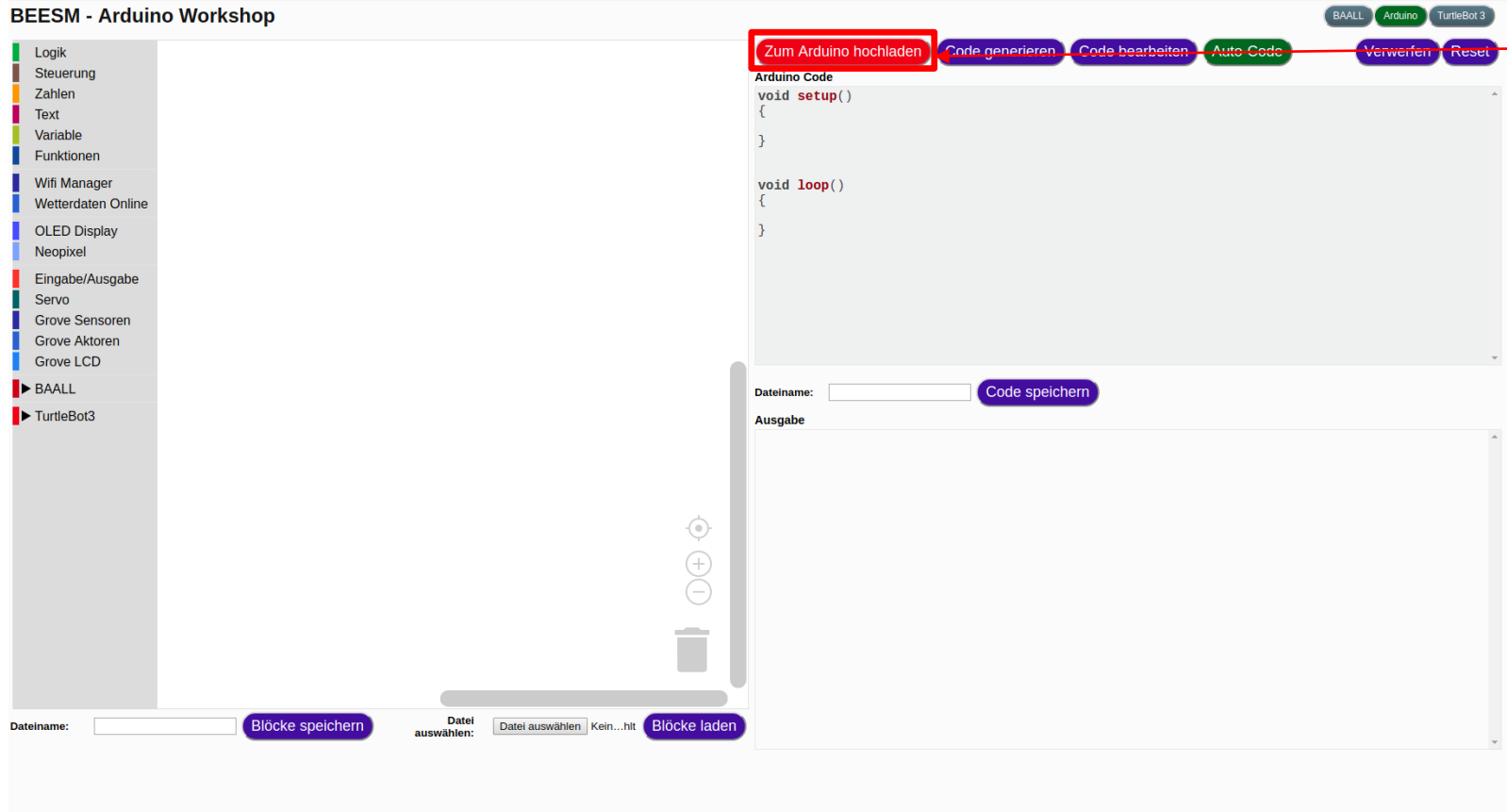
# Die Entwicklungsumgebung: BEEISM



Beim Klicken auf **Code generieren** wird der Arduino Code erstellt, den Ihr mit den Blöcken programmiert habt.

Tipp! Aktiviert **Auto-Code**, dann wird der Arduino Code mit jeder Änderung der Blöcke automatisch generiert.

# Die Entwicklungsumgebung: BEEISM



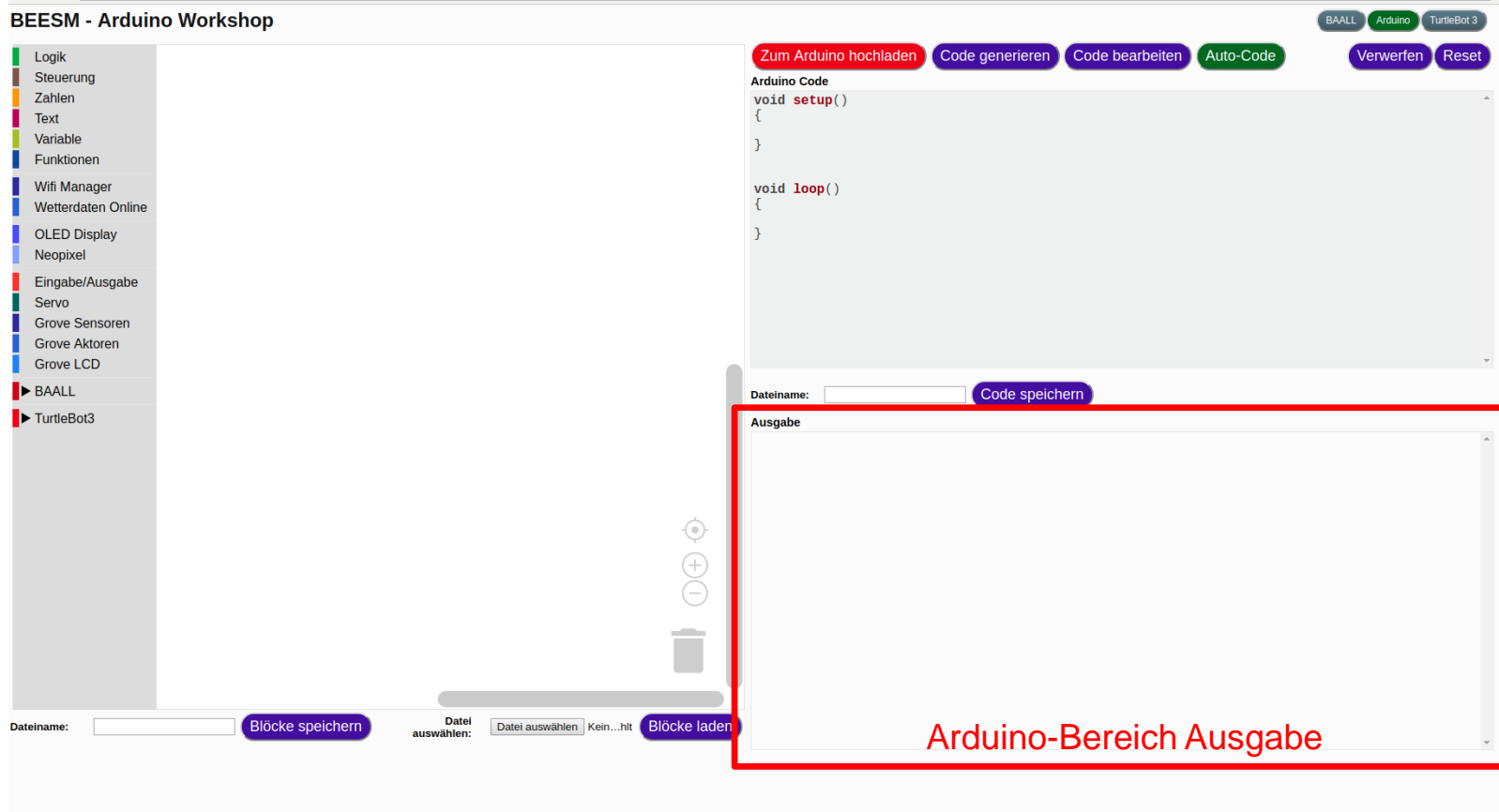
Mit einem Klick auf **Zum Arduino hochladen** wird das Programm auf den Arduino übertragen. Dies kann ein bisschen dauern. Also Geduld!

Wenn das Übertragen erfolgreich war, erscheint „Programm erfolgreich auf das Board hochgeladen“.

Wenn das Übertragen nicht erfolgreich war, erscheint „Fehler beim Hochladen des Programms“. Dann muss herausgefunden werden woran es lag. Evtl. seht Ihr im Ausgabe-Bereich eine Fehlermeldung.



# Die Entwicklungsumgebung: BEEISM



Im **Ausgabe Bereich** können wir sehen was unser Programm gerade macht. Also ob es uns z. B. einen Fehler ausgibt.

# Die Entwicklungsumgebung: BEESM




```
guest@smile-linux1: ~  
guest@smile-linux1:~$
```

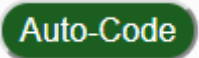


./start.sh



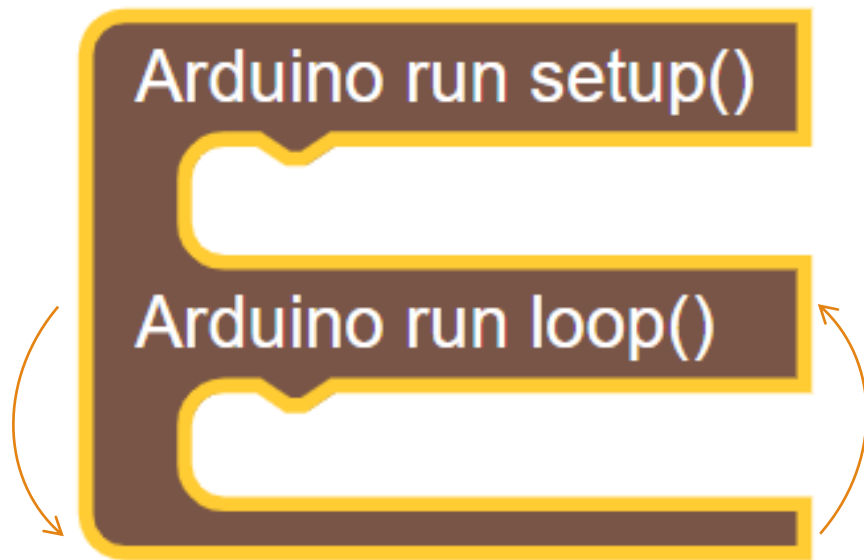
Danach Chrome öffnen

# Ein erstes Programm

- Gebe mit der Anweisung  „Halo Welt“ auf dem LC Display aus.


- Controller ist angeschlossen und BEESM gestartet? Dann:
- Schließ das LC Display an einen „I2C“ Steckplatz an.
- Aktiviere 
- Ziehe die Anweisungen für die LCD Textausgabe (siehe oben) in den **Programmierbereich**
- Lade dein Programm in den Controller hoch. 
- Der Upload dauert ein paar Sekunden. Warte bis die Bestätigung „Programm erfolgreich in den Controller hochgeladen“ erscheint. 

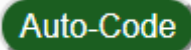
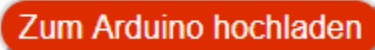

# Arduino Programmstruktur



- Setup:
  - Wird **einmal** zu Beginn des Programms ausgeführt.
- Loop:
  - Wird nach dem Setup als **Endlosschleife** ausgeführt

# Ein erstes Programm

- Gebe mit der Anweisung  „Halo Welt“ auf dem LC Display aus.

- Controller ist angeschlossen und BEESM gestartet? Dann:
- Schließ das LC Display an einen „I2C“ Steckplatz an.
- Aktiviere 
- **Ziehe den Block für die Arduino Programmstruktur in den Programmierbereich**
- Ziehe die Anweisungen für die LCD Textausgabe (siehe oben) in den **Setup** Bereich.
- Lade dein Programm in den Controller hoch. 
- Der Upload dauert ein paar Sekunden. Warte bis die Bestätigung „Programm erfolgreich in den Controller hochgeladen“ erscheint. 



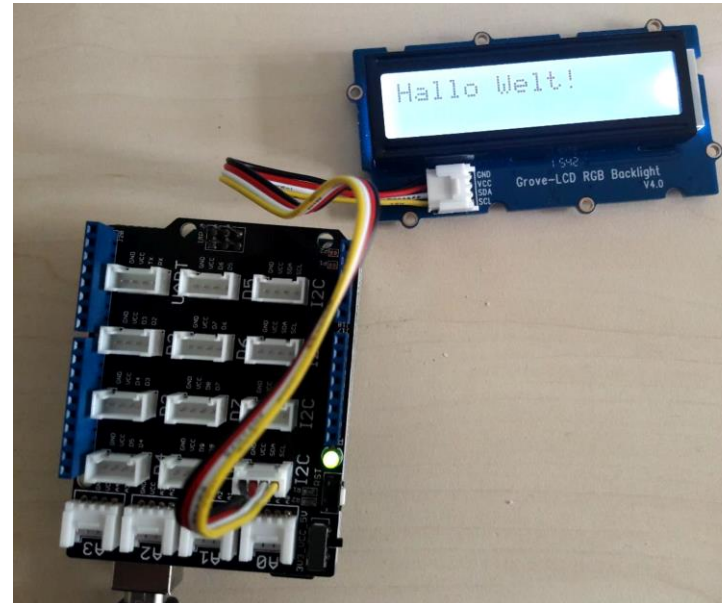
# Mein erstes Programm

Das Resultat sollte ungefähr so aussehen:

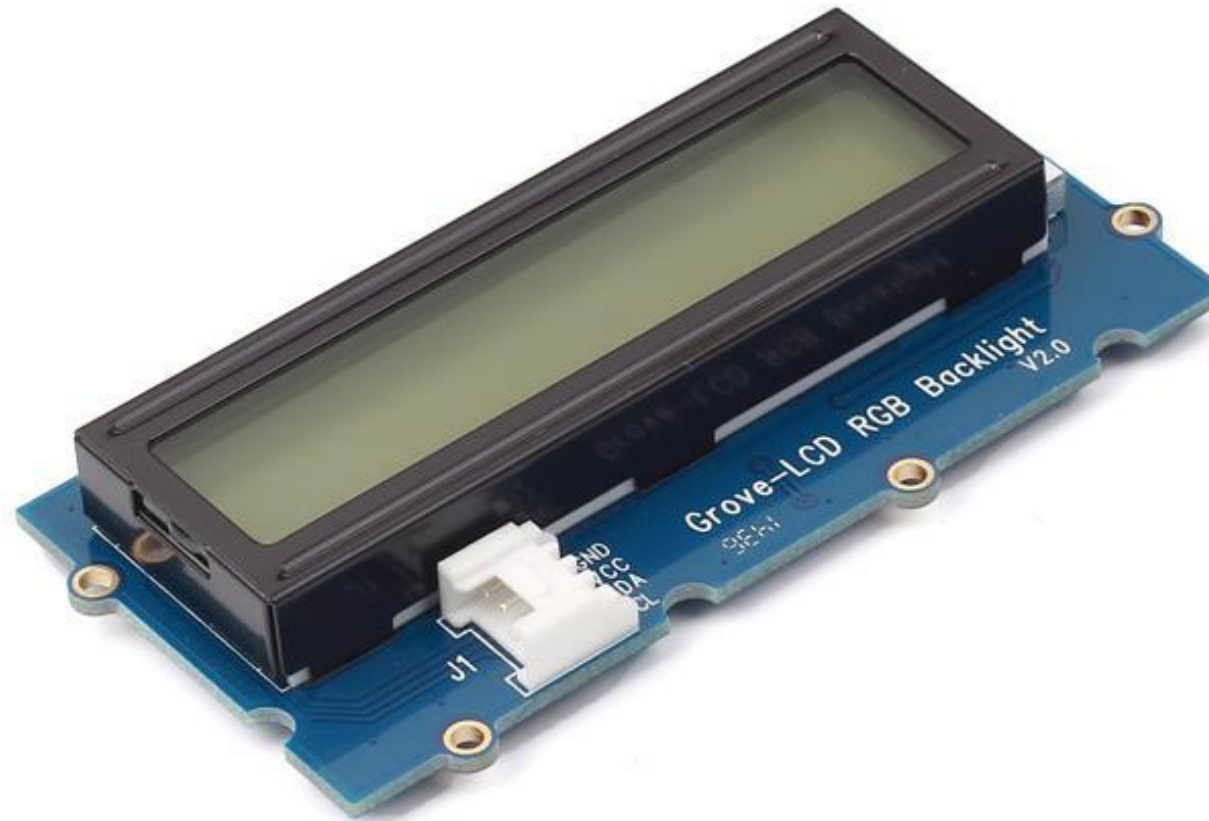
RGB LCD

Display anzeigen

“ Hallo Welt ”



# RGB LC Display




# Übungen: RGB LCD

- Recherchiert:
  - Wofür steht LCD?
  - Wofür steht RGB?
- Ändert euer Programm
  - Ändert den Text, den Ihr ausgeben
  - Verändert die Position des Textes
  - Wieviel Text passt auf das Display?
- Speichert das Programm ab.



# Übungen 2: RGB LCD

- Ändert euer Programm

- Gebe den Text in der **Loop** aus und ziehe einen Block für eine Pause  zusätzlich in die Loop.
- Ändert die Hintergrundfarbe. Gebt dazu für die Felder rot, grün, und blau jeweils **Werte zwischen 0 und 255** an.

Tipps!

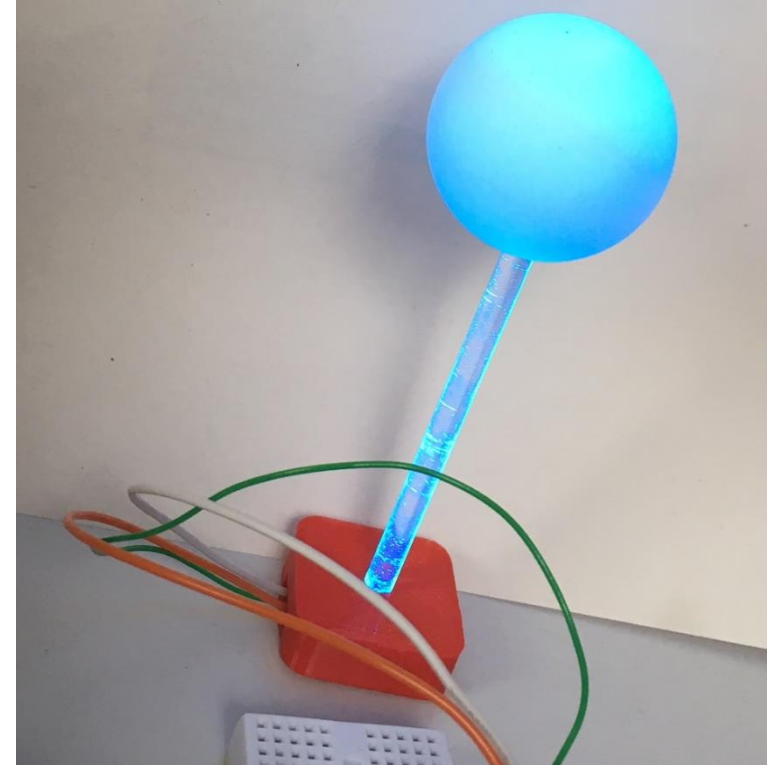
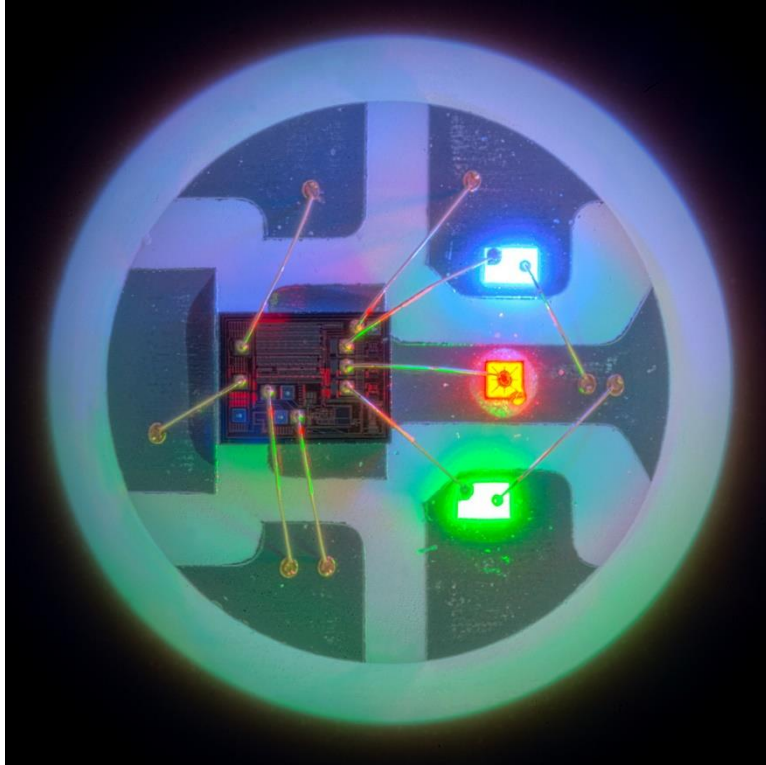
1000 Millisekunden entspricht 1 Sekunde

Den Block „delay“ findet ihr in der Blockpalette unter Steuerung

Frage 1: Was passiert, wenn Ihr den Text in der Loop ausgeben lasst? Warum? \_\_

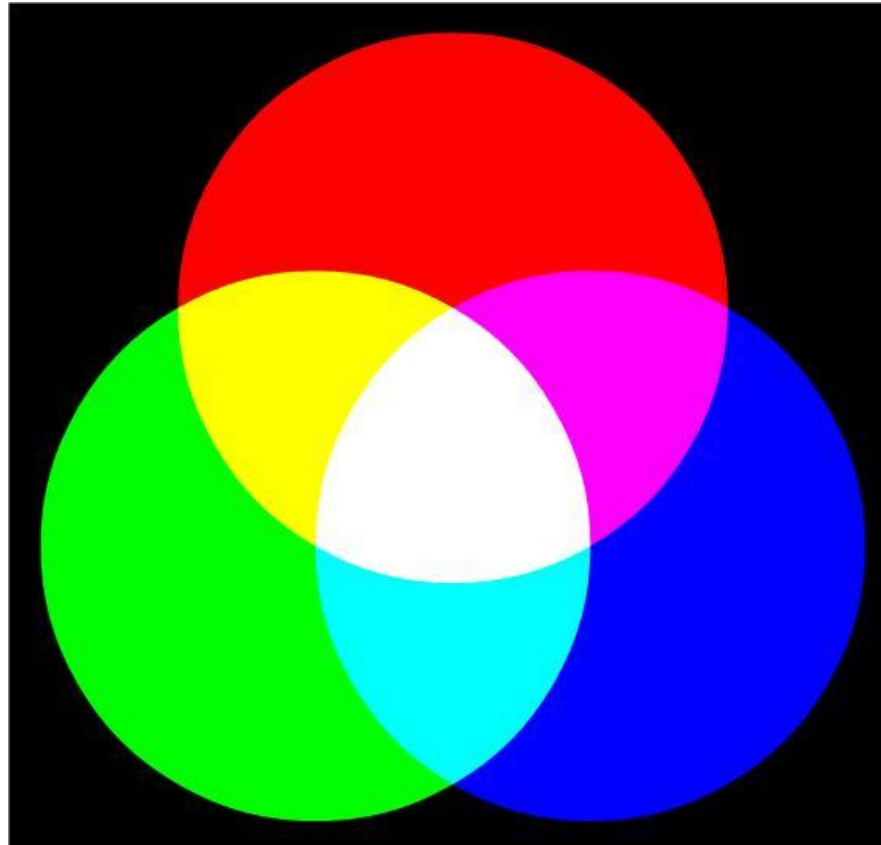
Frage 2: Was bedeutet die Zahl in dem „warte“ Block? \_\_

# RGB-LED





# Exkurs: Additives Farbmischmodell

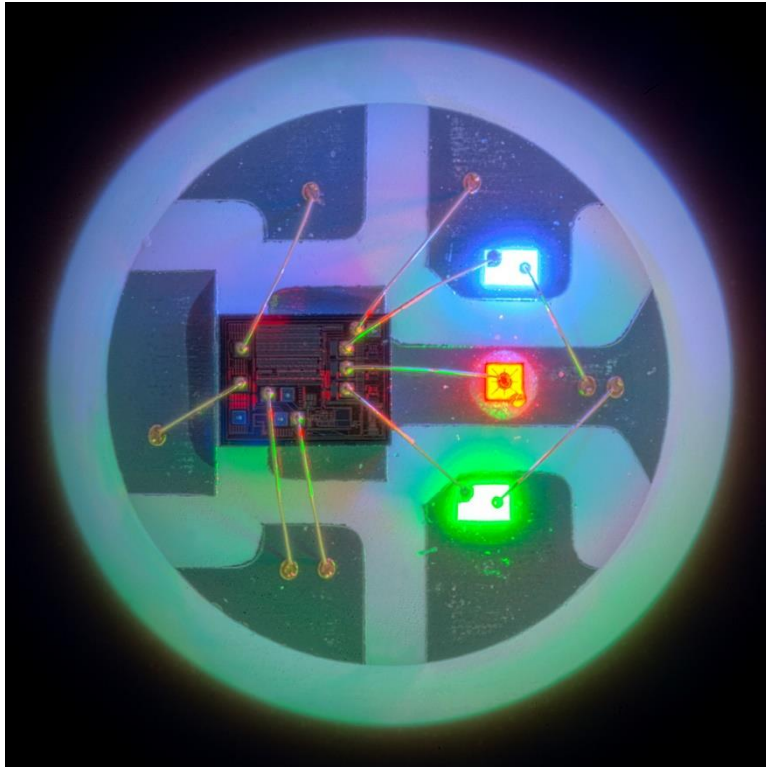


Alle modernen Monitore nutzen die additive Farbmischung im RGB-Farbmodell.

Bei der additiven Farbmischung werden durch Überlagerung der **drei primären Lichtfarben**, das ist **Rot**, **Grün** und **Blau**, viele weitere Farben gemischt.

Wie auf nebenstehender Abbildung zu sehen ist, entsteht im Ergebnis Weiß, wenn sich alle drei Grundfarben zu gleichen Anteilen überlagern. Addiert man nur die rote und grüne Strahlung, erhält man Gelb.

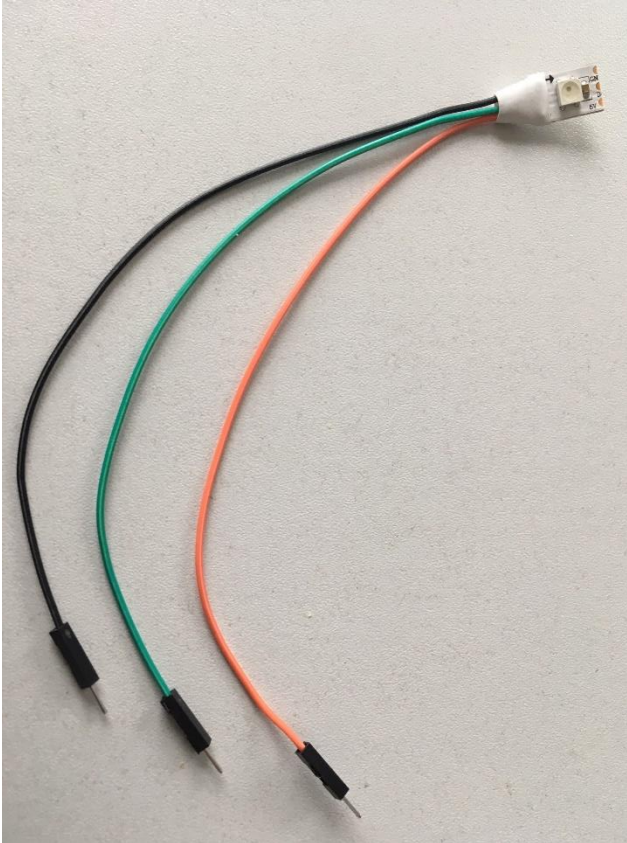
# RGB-LED



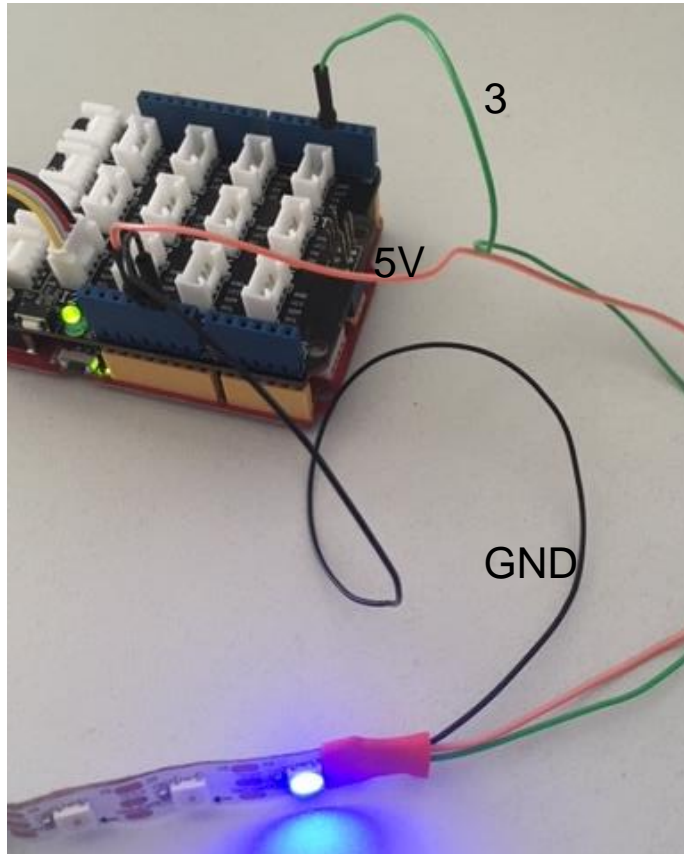
- Neopixel (→ „schlaue“ Pixel)
- In einem Pixel (LED-Lämpchen) steckt eine kleine rote, grüne und blaue LED, die sich einzeln ansteuern lassen.



# RGB-LED-Anschließen



# RGB-LED-Anschließen



— + Strom (5V)

— - Masse, Ground (G, GND)

— Daten, (D1...Dx)

# RGB-LED-Neopixel

Neopixel anmelden pin# 1 Anzahl Pixel 0 Typ NEO\_GRB NEO\_KHZ800

Helligkeit setzen pin# 1

Farbe setzen pin# 1 Pixelnummer 0 rot 0 grün 0 blau 0

**Anmelden:** Wir definieren den **Pin**, an welchen wir die Datenleitung (gelbes, grünes oder blaues Kabel) unserer LED-Lämpchen angeschlossen haben und **wieviele LEDs** wir in Reihe geschaltet haben. Dies geschieht 1mal im **Setup** des Programms.

**Optional:** Wir können die **Helligkeit** der Lämpchen setzen. Ohne Angabe leuchten die Lämpchen mit voller Intensität, das ist gleich 255. Diese Angabe sollte nur einmal am Anfang des Programms im **Setup** gesetzt werden, wenn wir die Lämpchen insgesamt weniger hell haben wollen.

**Farbe setzen:** Hier sagen wir dem Programm, in welcher Farbe das LED-Lämpchen leuchten soll. Die Farbe wird aus den Werten für rot, grün und blau gemischt. Die Werte können jeweils zwischen 0 (aus) und 255 (höchste Intensität) liegen. **Pixelnummer** 0 bedeutet, dass wir das erste Lämpchen einschalten.

Die oben gesetzten Farben muss ich nun an die LEDs senden. Sonst passiert nichts. Diese Anweisung brauchen wir nach jeder Änderung der Lämpchen.

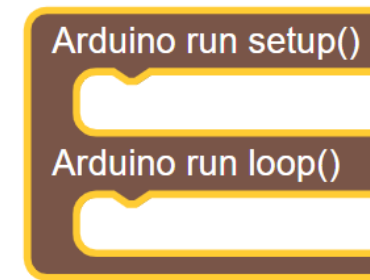
Wichtig! Bei allen Blöcken muss die **Pin** angegeben werden, an welchem wir die LED angeschlossen haben

Neopixel anzeigen pin# 1

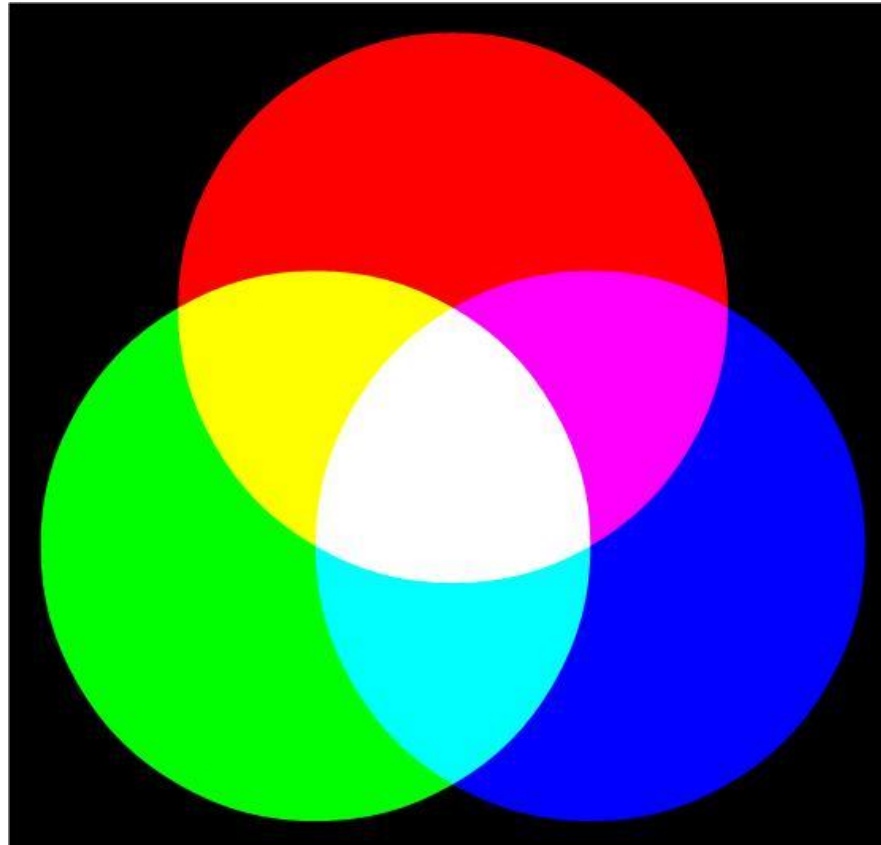


# Ein weiteres Programm

- Schreibe ein Programm, dass das Lämpchen in deiner Wunschfarbe leuchten lässt.
- Beachte dabei folgende Schritte:
  - Schließt das LED Lämpchen an den Controller
  - Ziehe den Block für die Arduino Programmstruktur in den Programmierbereich
  - Wir müssen die LEDs im Setup **anmelden** und dann mit **Farbe setzen** und **anzeigen** zum leuchten bringen. Die passenden Befehle findet Ihr unter Neopixel.
  - Ladet euer Programm in den Controller hoch



# Arbeitsblatt Neopixel & Challenge



Alle modernen Monitore nutzen die additive Farbmischung im RGB-Farbmodell.

Bei der additiven Farbmischung werden durch Überlagerung der **drei primären Lichtfarben**, das ist **Rot**, **Grün** und **Blau**, viele weitere Farben gemischt.

Wie auf nebenstehender Abbildung zu sehen ist, entsteht im Ergebnis Weiß, wenn sich alle drei Grundfarben zu gleichen Anteilen überlagern. Addiert man nur die rote und grüne Strahlung, erhält man Gelb.

# Was ist ein Programm?

- Habt Ihr da Ideen?

# Sensoren/Aktoren?



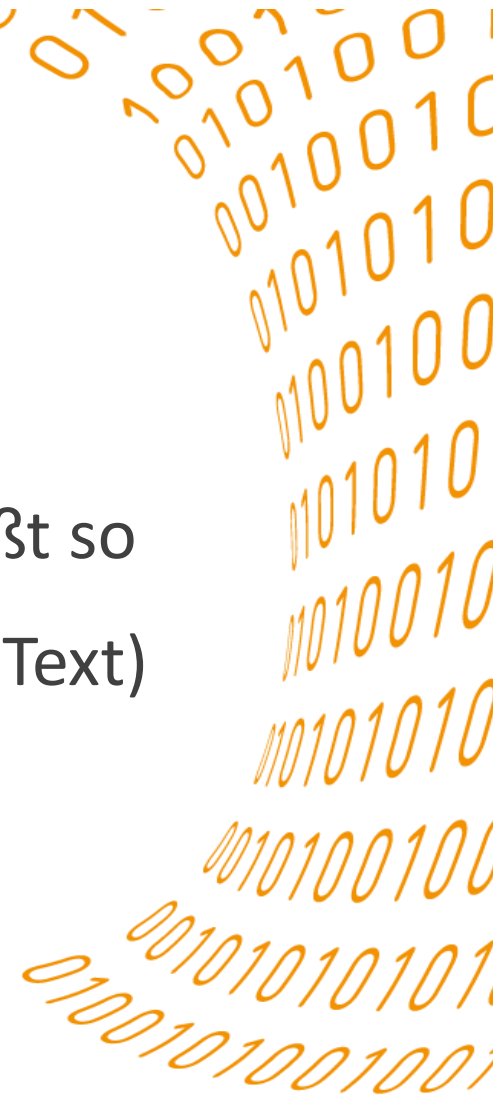
# Hands-On Arduino



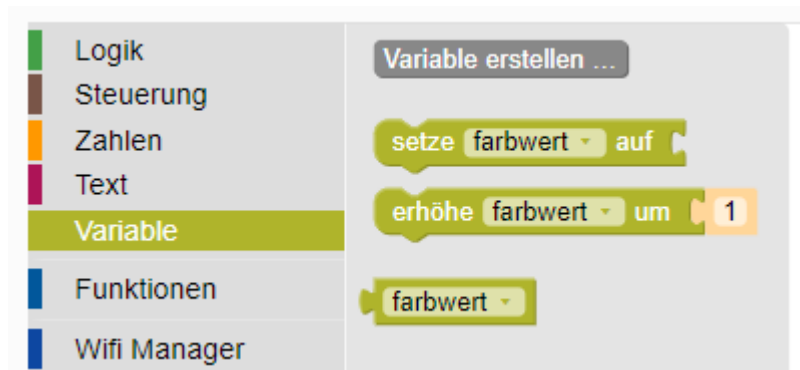
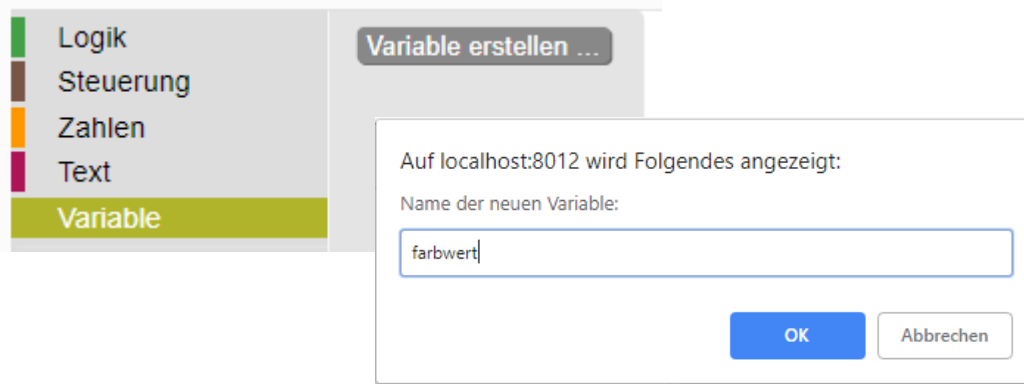


# Einführung: Variablen

- Eine Variable ist ein Platz, um Werte zu speichern
- Ähnlich wie ein Karton, in den ich etwas hinein tun kann
- Dieser Karton hat einen eindeutigen Namen, also nur er heißt so
- Bei BEESM können Variablen **NUR** Zahlen enthalten (keinen Text)



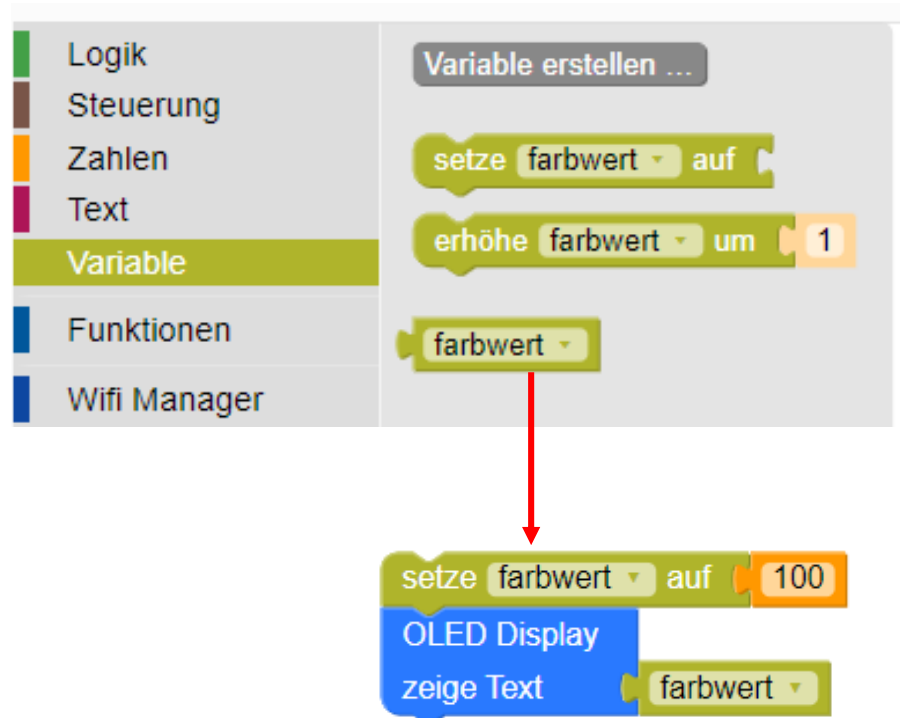
# Variablen anlegen



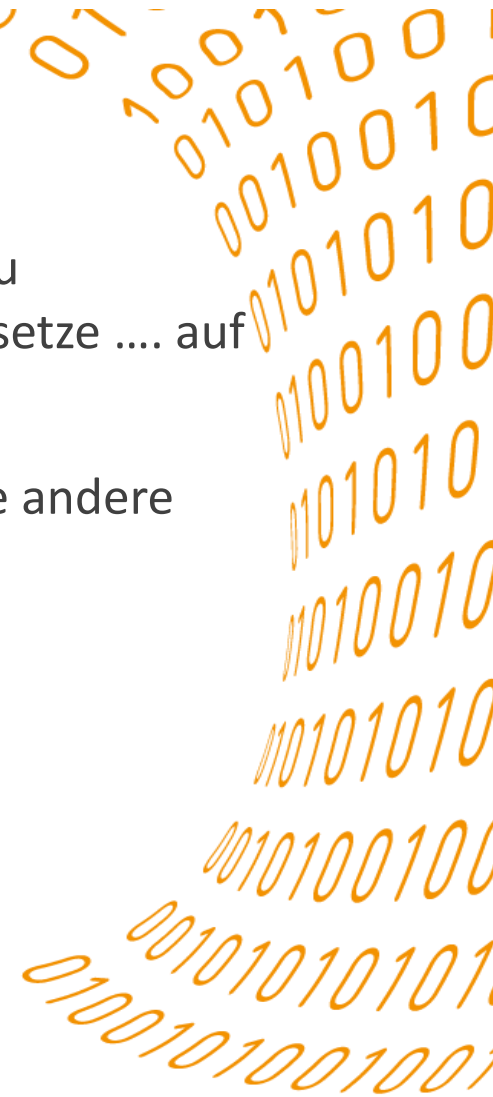
- Erstelle eine Variable im Punkt „Variable“
- Dabei muss der Variable ein Name gegeben werden.
- Danach erscheinen neue Befehle unter „Variable“ für diese Variable



# Variablen anlegen

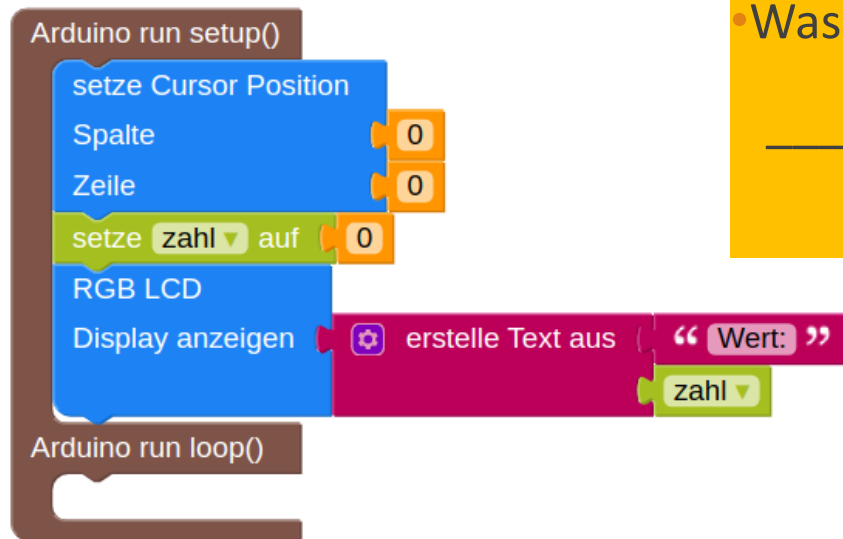


- Um einen Wert in einer Variablen zu speichern, nehmen wir den Block „setze .... auf x“.
- Zum Lesen zieht die Variable in eine andere Operation rein.



# Übungen: Variablen

Erstellt eine Variable, speichert einen Wert (Zahl) in der Variablen und lässt den Wert der Variablen auf dem Display ausgeben!



• Was gibt der Code auf dem Display aus?

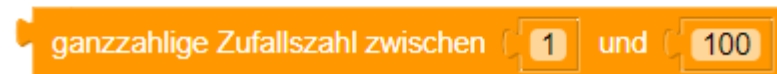
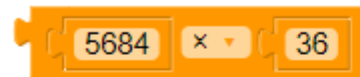
\_\_\_\_\_

# Übung 1: Variablen

Nun lassen wir den Computer rechnen.

Anstelle einer einzigen Zahl wie im vorherigen setzen wir die Variable auf komplexere Terme („mathematische Gebilde“). Entsprechende Blöcke findet ihr unter Zahlen.

Zum Beispiel folgende:



# Übung 2: Variablen

- Ändert euer Programm:

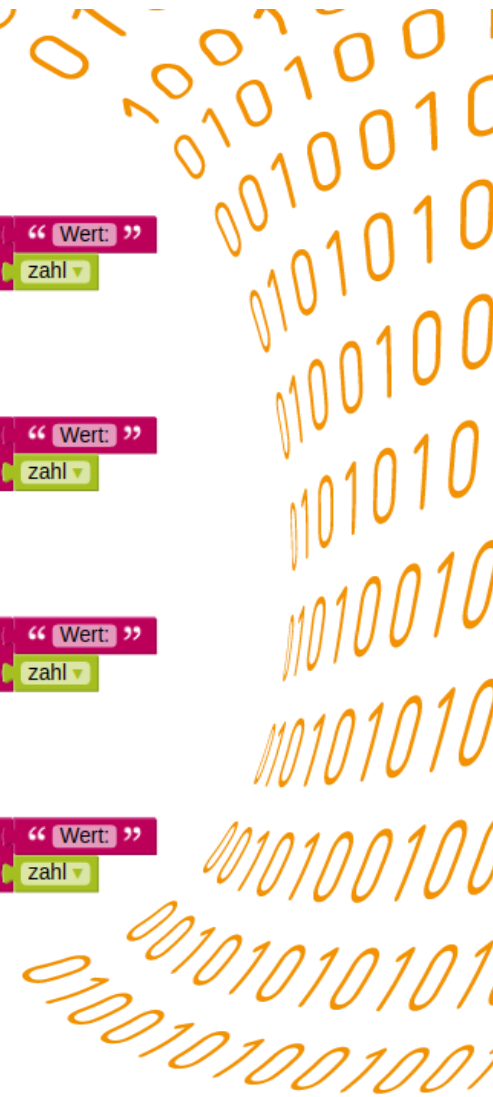
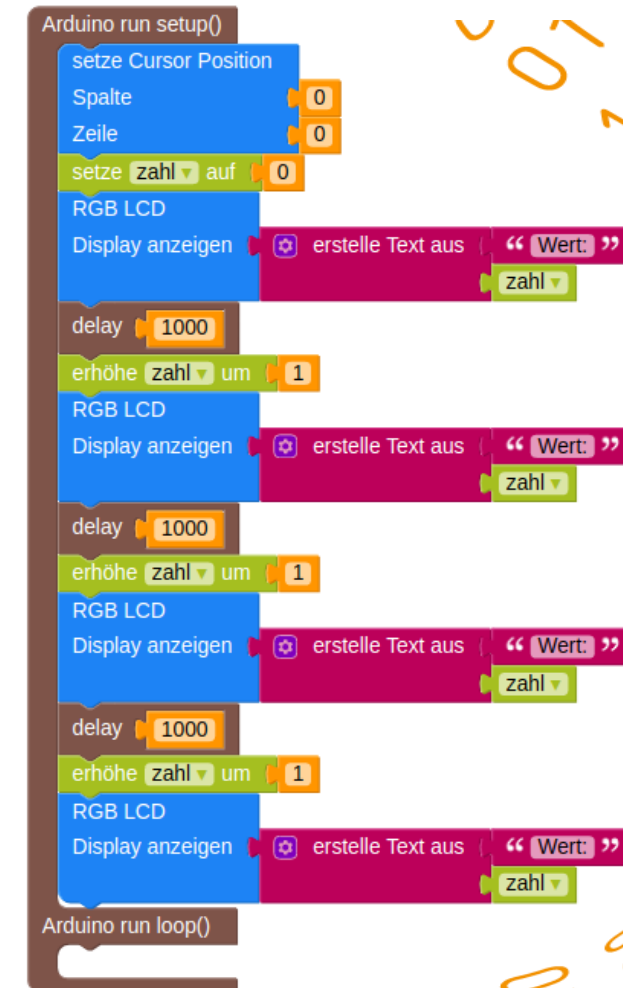
- Warte nach der Ausgabe der Variable für eine Sekunde
- Erhöhe dann den Wert der Variable um eins
- Gebe die geänderte Variable aus
- Wiederhole dieses drei mal






# Lösung: Variablen 2

- Ändert euer Programm:
  - Warte nach der Ausgabe der Variable für eine Sekunde
  - Erhöhe dann den Wert der Variable um eins
  - Gebe die geänderte Variable aus
  - Wiederhole dieses drei mal



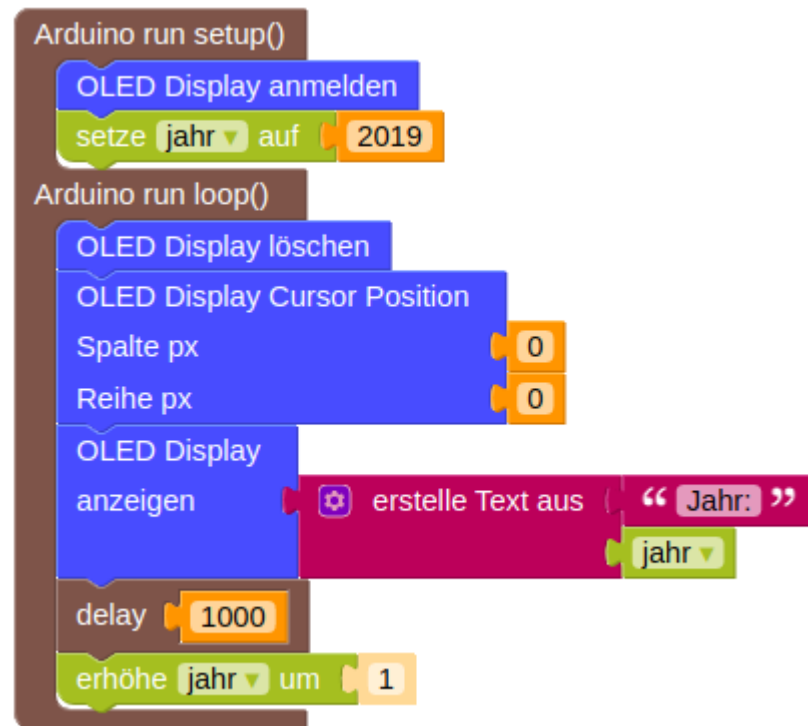
# Übung 3: Variablen

- Erweitert euer Programm und programmiert einen Zähler:
  - Wartet nach der Ausgabe der Variable für eine Sekunde
  - Erhöht dann den Wert der Variable um eins 
  - Gebt die geänderte Variable aus
  - Diese schritte sollen unendlich oft wiederholt werden

Tipp!  
Programmiert hierzu in der **loop()**

# Lösung: Variablen 2

- Ändert euer Programm: Zähler



# Sensoren

Wo schließe ich den Sensor an?

Wie lese ich den Sensor aus?

Welche Werte gibt der Sensor aus?

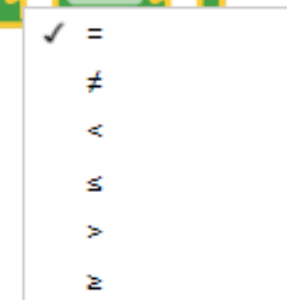
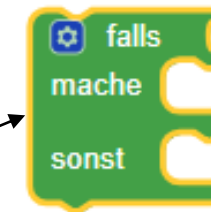
Teste die Ausgaben unter unterschiedlichen Bedingungen!

Kennst du Beispiele aus deinem Alltag, wo dieser Sensor gebraucht wird?

# Bedingungen

- Bedingungen dienen zum Steuern des Programmflusses. Sie sind so etwas wie „Falls dies zutrifft mache dies, sonst das“.

- Dazu benötigen wir einen **Bedingungsblock** ...
- ...und einen **Vergleichsblock**. Es gibt einfache Vergleiche, wie „kleiner“, „größer“ und „gleich“
- Damit überprüfen wir, ob der Vergleich zweier Werte wahr ist oder falsch. Ist  $1 = 1$ ? Ist er wahr, werden die Anweisungen hinter „mache“ ausgeführt.
- Beide Blöcke findet ihr unter „Logik“



# Übung Bedingungen

- Ändert Euer Sensor-Programm ab:
  - Überprüft mit einer Bedingung, ob der Wert eures Sensors größer eines bestimmten Wertes x ist.
  - Falls der Wert größer x ist, dann ändert die Farbe des LC-Displays auf grün.
  - Falls der Wert kleiner x ist, ändert die Farbe des LC-Displays auf eine andere Farbe, z.B. rot.

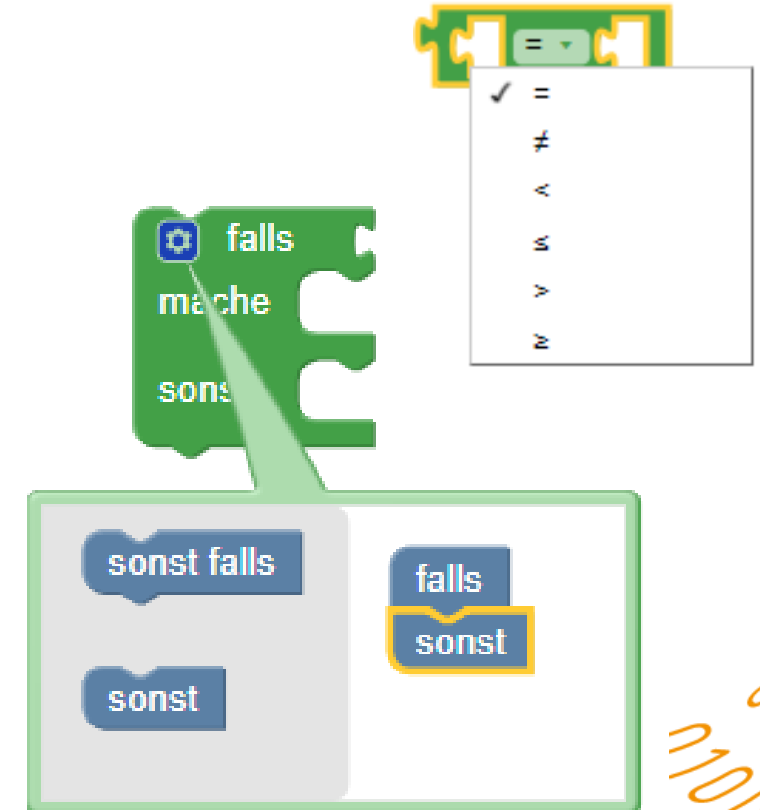
Lichtsensord; x=400

Soundsensord; x=600

Temperatursensord; x=23

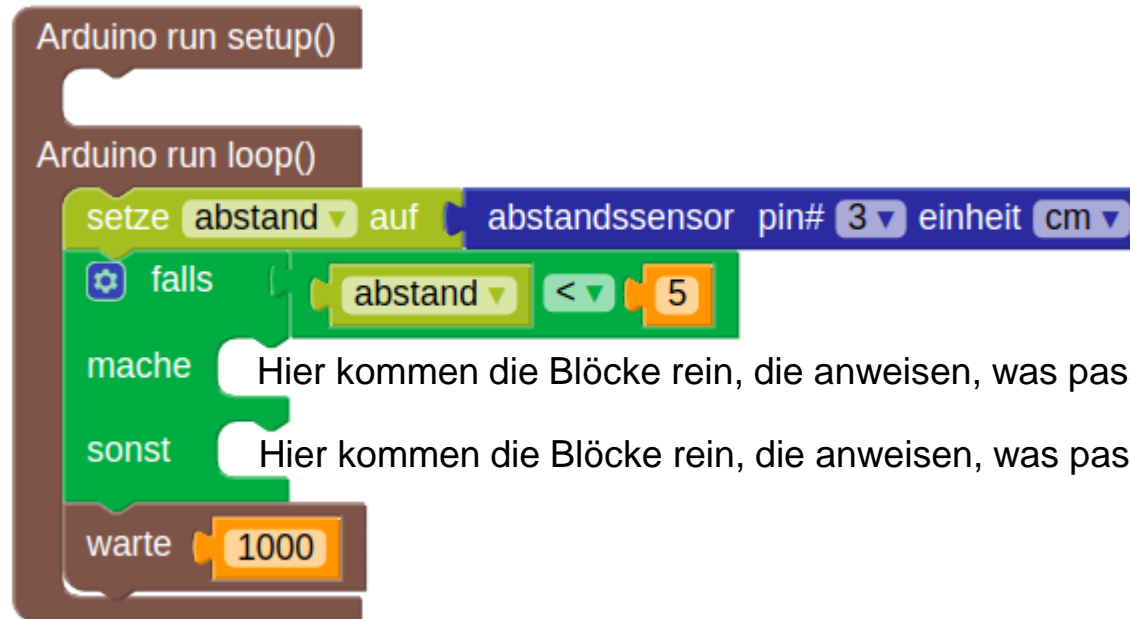
Distanzsensord, x=10

PIR, x=0





# Bedingungen



**mache** Hier kommen die Blöcke rein, die anweisen, was passiert, **wenn der Abstand kleiner als 5 cm ist**

**sonst** Hier kommen die Blöcke rein, die anweisen, was passiert, **wenn der Abstand nicht kleiner als 5 cm ist**

