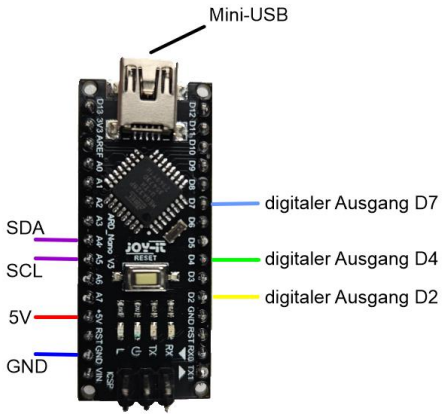
Escape-Würfel	**Teil 1**
Grundlagen der Elektronik und Programmierung	

1. Kurze Vorbemerkung

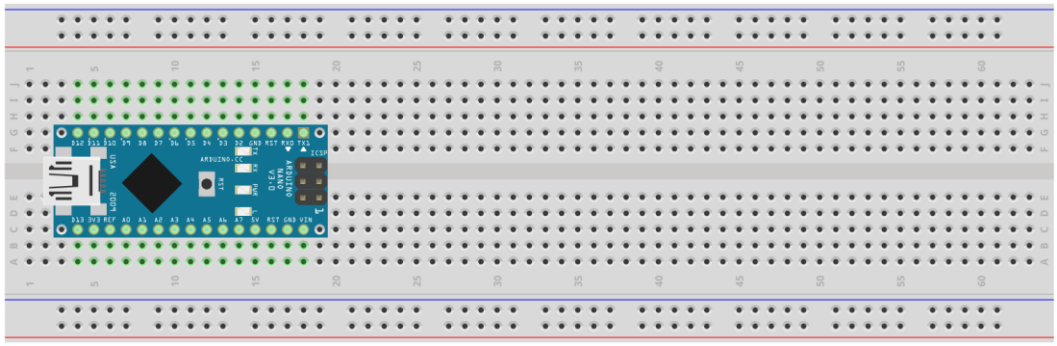
   Sämtliche Abbildungen mit Breadboarddarstellungen sind mit Hilfe der Software *fritzing* erstellt worden (https://fritzing.org).

2. Der Mikrocontroller und die Arduino IDE

   Im Projekt wird ein Mikrocontroller von joy-it verwendet, der vollständig kompatibel zu einem Arduino Nano V3 ist. Die folgende Abbildung zeigt, wo die PINs zu finden sind, die wir für unsere Schaltung benötigen:



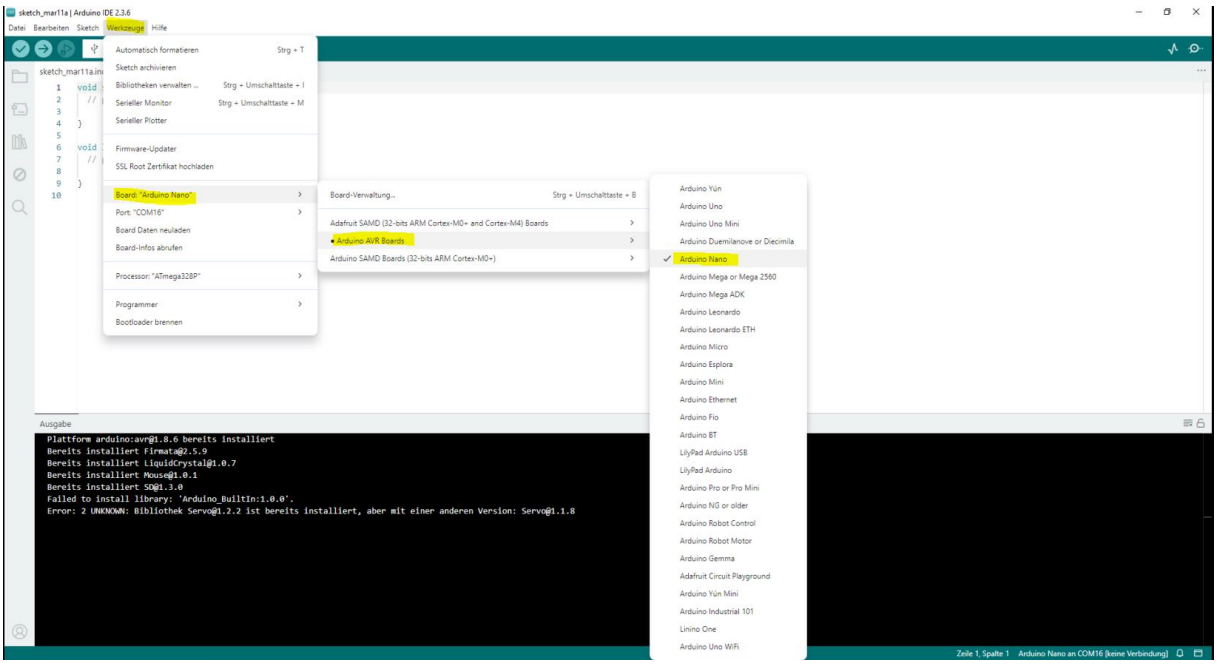
- **Stecke** den Mikrocontroller auf die Steckplatte **und verbinde** ihn über das USB-Kabel mit dem Computer.



- **Öffne** die Programmierumgebung für den Mikrocontroller, die sogenannte Arduino IDE. Das geht über einen Doppelklick auf das folgende Icon auf dem Desktop des Computers:

Damit der Mikrocontroller und die Arduino IDE miteinander kommunizieren können, müssen einige Grundeinstellungen vorgenommen werden.

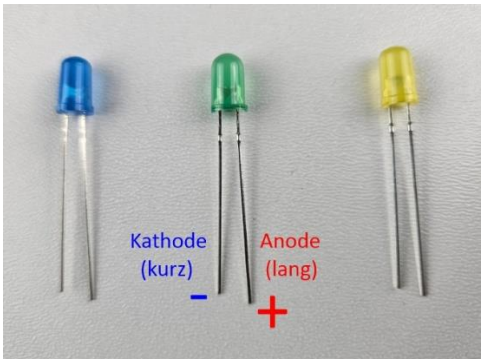
- **Wähle** unter „Werkzeuge“ und „Board“ den „Arduino Nano“ **aus**:



- **Wähle** ebenfalls unter „Werkzeuge“ und „Port“ den richtigen Port **aus**. Er lässt sich daran erkennen, dass hinter dem Eintrag „COM“ und zugehöriger Nummer in Klammern das Arduinoboard genannt wird.

3. <u>Eine erste Schaltung mit einer Leuchtdiode auf dem Steckbrett</u>

Im benötigten Material für das Würfelprojekt befinden sich drei farbige Leuchtdioden:



Leuchtdioden werden auch kurz mit „LED“ bezeichnet, was für für „**l**ight-**e**mitting **d**iode“ steht und somit schon zwei Eigenschaften ausdrückt:

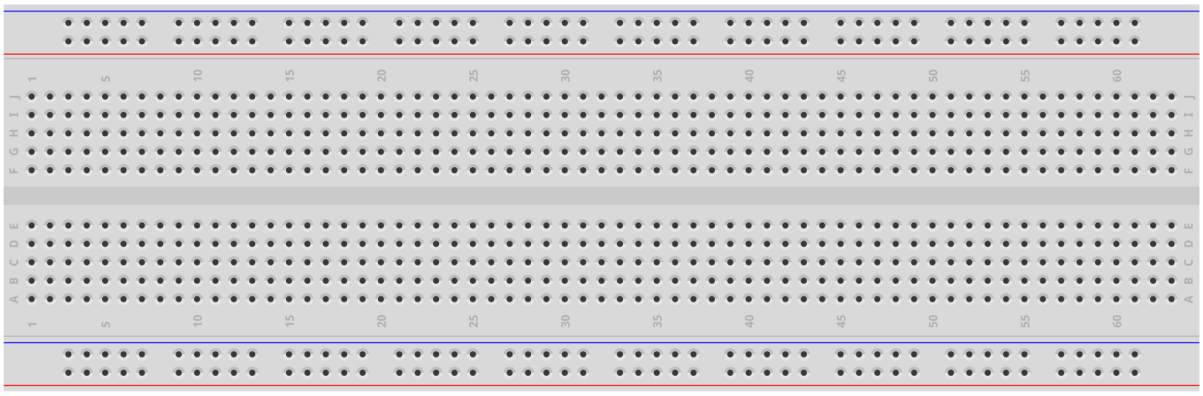
i. Diese Halbleiterbauteile sind Dioden, d.h. sie haben eine Durchlassrichtung und eine Sperrrichtung für den elektrischen Strom.

ii. Werden sie von einem elektrischen Strom geeigneter Stromstärke in Durchlassrichtung durchströmt, dann leuchten sie in einer vom konkreten Bauteil vorgegebenen Farbe, bei uns in blau, gelb oder grün.

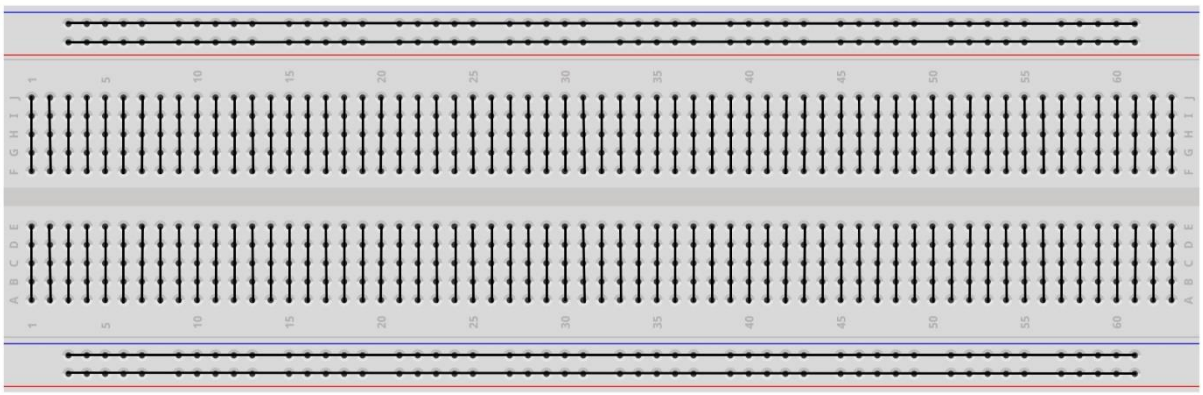
Das Bild oben zeigt die Durchlassrichtung, bei der die Anode (längeres Beinchen) an „+“ und die Kathode (kürzeres Beinchen) an „-“ anliegen muss.

Unser Mikrocontroller stellt uns an den digitalen Ausgängen Spannungen in Höhe von 5V zur Verfügung, die wir zum Betreiben der LEDs nutzen wollen. Das ist mehr als die von uns genutzten LEDs vertragen, denn der von dieser Spannung hervorgerufene elektrische Strom hätte eine so große Stromstärke, dass die Leuchtdioden zerstört würden. Daher muss für jede LED ein geeigneter elektrischer Widerstand in Reihe mit ihr geschaltet werden, an dem ein Teil der Spannung abfällt. Für die von uns genutzten LEDs zeigen die Datenblätter, dass dafür ein Widerstand in Höhe von 150Ω benötigt wird. Der Farbcode bei den hier verwendeten Kohleschicht-Widerständen für 150Ω ist „braun-grün-braun“ (und ein goldener Ring, der die Info zur Toleranz des Wertes angibt).

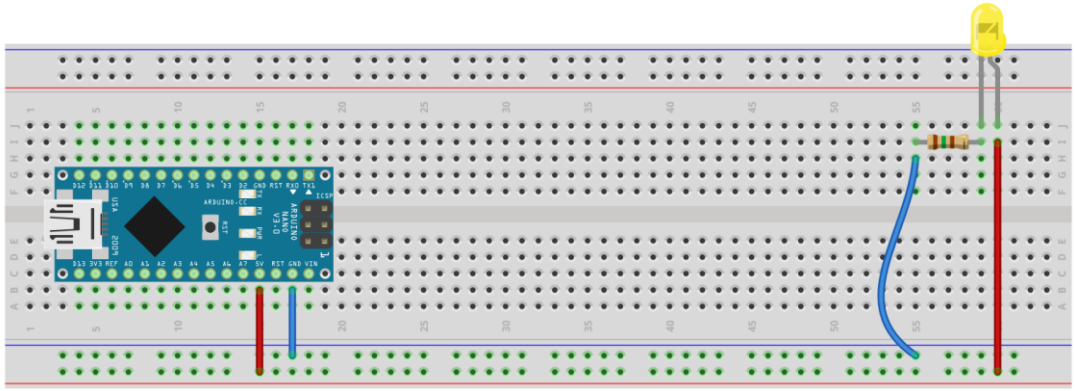
Auch wenn unsere Gesamtschaltung später dauerhaft gelötet in den Würfel eingebaut werden soll, ist es für die Experimentierphase viel komfortabler, die Schaltungen zunächst auf einer Steckplatine („Breadboard“) aufzubauen:



Die einzelnen Steckplätze sind dabei innerhalb des Breadboards nach folgendem Schema elektrisch verbunden:

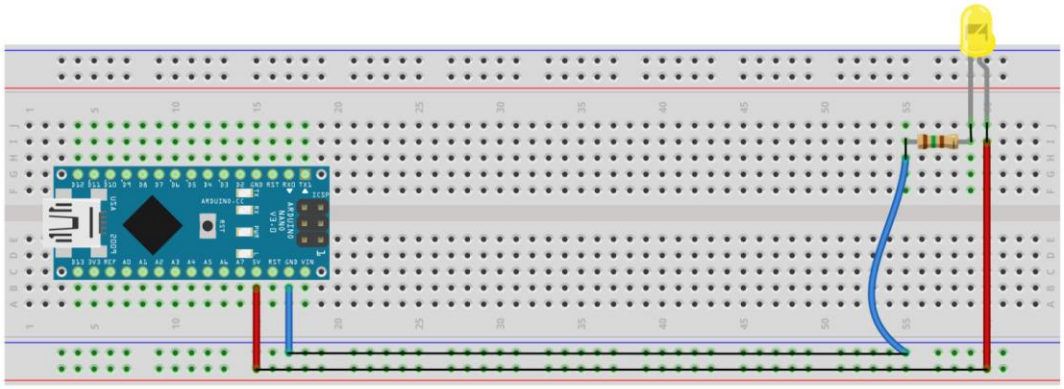


Die Reihenschaltung von Leuchtdiode und Widerstand lässt sich zum Beispiel in dieser Form realisieren:



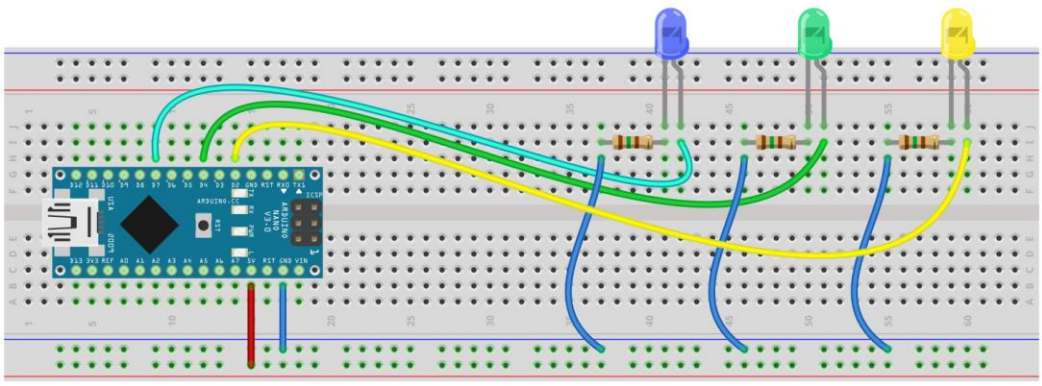
- **Baue** diese Schaltung auf dem Breadboard **auf**.

Die elektrischen Kontakte bestehen dann konkret in dieser Form:



Die LED sollte dauerhaft leuchten, sobald der Arduino über das USB-Kabel mit dem Rechner verbunden ist. Ist das nicht der Fall, dann kann es sein, dass die LED nicht in der korrekten Polung eingesteckt ist. Dreh sie einfach mal um, dann ist das Problem oft schon behoben. Ansonsten kontrolliere nochmal alle einzelnen Steckkontakte.
Bemerkung: Der Minuspol der Schaltung ist hier mit „GND“ bezeichnet, was für „ground“ steht. In deutscher Sprache sagt man dazu auch „Erde“ und meint das Bezugsniveau, auf das sich Spannungen beziehen.

- **Baue** jetzt auf dem Breadboard die folgende Schaltung **auf**:



4. Eine LED mit dem Arduino steuern

Bisher wurde der Arduino nur als „Steckdose“ für die gelbe LED benutzt, d.h. er hat dauerhaft die Versorgungsspannung von 5V zur Verfügung gestellt. Jetzt soll das Verhalten der LED mit Hilfe geeigneter Programme (werden bei Arduino auch „Sketch“ genannt) gezielt gesteuert werden.

- **Öffne** in der Arduino IDE unter „Datei“ und „Neuer Sketch“ einen neuen Sketch.

Dieser wird zunächst einmal so aussehen:

sketch_mar11a.ino

```
1  void setup() {
2    // put your setup code here, to run once:
3
4  }
5
6  void loop() {
7    // put your main code here, to run repeatedly:
8
9  }
10
```

Auch wenn noch nicht wirklich Programmcode existiert, machen diese wenigen Zeilen schon eine Grundstruktur eines solchen Sketches deutlich:

i. Es gibt einen Bereich „`setup`“, der innerhalb der geschweiften Klammern Befehle aufnehmen kann. Diese werden zu Beginn des Programms ein einziges Mal in der vorgegebenen Reihenfolge abgearbeitet.
ii. Es gibt einen zweiten Bereich „`loop`“, der ebenfalls einen solchen Bereich innerhalb von geschweiften Klammern aufweist. Auch hier werden die eingefügten Befehle der Reihe nach abgearbeitet, mit dem Unterschied, dass der loop nach dem letzten Befehl wieder von vorne anfängt. Die Bezeichnung „`loop`“ verdeutlicht das bereits, denn es handelt sich in der Tat um eine (Dauer-)“Schleife“.
iii. Zwei Schrägstriche (`//`) hintereinander beginnen einen Kommentar, der Informationen zur Programmierung enthält, die vom Arduino jedoch ignoriert werden. Der Kommentar macht ein Programm für andere Personen aber häufig besser verständlich.

Wir nutzen den digitalen PIN D2 als digitalen Ausgang für die gelbe LED. Das kann man sich so wie eine schaltbare Steckdose vorstellen. Schaltet man ihn mit einem passenden Befehl ein, dann ist die Steckdose eingeschaltet und weist 5V Spannung gegenüber dem Bezugsniveau „GND“ auf. Ist sie dagegen ausgeschaltet, dann beträgt die Spannung gegenüber GND null, d.h. der PIN liegt selbst auf diesem Niveau.
Der eben geöffnete neue Sketch kann jetzt mit einigen Befehlen zu einem ersten sinnvollen Programm ergänzt werden:

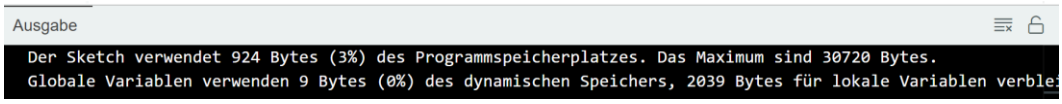
```
void setup() {
  pinMode(2, OUTPUT);
}

void loop() {
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```

Im Setup wird mit dem Befehl „`pinMode(2, OUTPUT)`“ festgelegt, dass der digitale Pin mit der Nummer 2 als Ausgang genutzt wird. Das heißt, dass der Pin Informationen in Form von Spannung an das angeschlossene Bauteil ausgibt. Wir werden in wenigen Abschnitten kennenlernen, dass digitale Pins auch Eingänge (also Empfänger von Informationen) sein können, so dass diese Festlegung notwendig ist. Die Befehlszeile wird mit einem Semikolon abgeschlossen, was bei allen folgenden Befehlszeilen ebenfalls erforderlich ist.
In der anschließenden Hauptschleife wird der digitale Pin 2 zunächst mit dem Befehl „`digitalWrite(2, HIGH)`“ eingeschaltet, dann wird mit dem Befehl „`delay(1000)`“ für eine Verzögerung von 1000 Millisekunden (also von einer Sekunde) gesorgt. In der Wartezeit leuchtet die LED, im Anschluss wird sie mit dem Befehl „`digitalWrite(2, LOW)`“ wieder ausgeschaltet und bleibt wegen der erneuten Verzögerung eine Sekunde aus. Dann beginnt die Schleife wieder von vorne. Die gelbe LED wird also als Blinklicht betrieben, das immer abwechselnd eine Sekunde an und eine Sekunde aus ist.

Soweit die Theorie. Damit die gelbe LED das gewünschte Blinkverhalten auch in der Realität zeigt, muss das Programm auf den Arduino hochgeladen werden. Das geschieht in zwei Schritten: In einem ersten Schritt muss ein Compiler das Programm in einen für den Arduino lesbaren Code (Hex-Datei) überführen, dann kann es in einem zweiten Schritt auf den Mikrocontroller übertragen werden.

- **Ergänze den Sketch** wie oben beschrieben.
- **Starte den Hochladevorgang** durch einen Klick auf den „Rechtspfeil“ in der Arduino IDE oben links:

- **Beobachte** die Ausgaben während dieses Vorgangs im Ausgabefenster unten in der Arduino IDE:



  Wenn alles gut läuft, dann sollten zunächst beim Compilieren einige Informationen zum Sketch erfolgen und dann der Hochladevorgang erfolgreich durchgeführt werden.
  Andernfalls liegt eventuell eines der folgenden sehr häufigen Probleme vor:
  - In der Arduino IDE wurde entweder nicht das richtige Bord eingestellt oder nicht der richtige Port ausgewählt (siehe oben; bei Bedarf kontrollieren und gegebenenfalls korrigieren).
  - Die USB-Verbindung ist nicht hergestellt (Kabelverbindung kontrollieren).
  - Es wurde ein Befehl falsch geschrieben, bei der Klammersetzung sind Fehler aufgetreten oder es wurden nicht alle Befehlszeilen mit einem Semikolon abgeschlossen. Diese Fehler werden nach dem Kompilieren häufig in der Ausgabe unten angezeigt.

- **Ändere** das Programm so **ab**, dass die blaue LED statt der gelben LED blinkt und dabei jeweils nur eine halbe Sekunde an und eine halbe Sekunde aus ist.

5. Integer-Variablen

Die Änderungen im letzten Abschnitt sind hoffentlich gut gelungen, waren aber lästiger als notwendig. Sowohl für den anderen LED-Pin als auch für die neue Wartezeit mussten jeweils zwei Einträge im Programmtext geändert werden.

Das geht besser:

```
int LEDPin = 2;
int wartezeit = 1000;

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

void loop() {
  digitalWrite(LEDPin, HIGH);
  delay(wartezeit);
  digitalWrite(LEDPin, LOW);
  delay(wartezeit);
}
```

In den beiden ersten Zeilen werden die beiden „Variablen“ mit den Namen „`LEDPin`“ und „`wartezeit`“ eingeführt („deklariert“). Das geschieht hier vor Setup und Loop und gilt damit für das ganze Programm. Beachte, dass in der Arduino IDE Groß- und Kleinschreibung unterschieden werden.
Variablen kann man sich wie Kartons vorstellen, die beschriftet sind und in die Dinge hineingelegt werden können. Im konkreten Fall steht auf den beiden Kartons „LEDPin“ und „wartezeit“ und in jeden Karton kann eine beliebige ganze Zahl (aus dem Bereich von ca. -32000 bis 32000) gelegt werden, denn beide Variablen sind vom Typ „Integer“. Das wird durch das vorangestellte „`int`“ festgelegt. Mit dem Gleichheitszeichen wird beiden Variablen ein erster Wert zugeschrieben. Wenn im Programm irgendwo der Name einer Variablen aufgerufen wird, arbeitet das Programm stattdessen mit dem hinterlegten Wert, d.h. die Variable ist quasi Platzhalter für diesen Wert. Der Wert kann später auch geändert werden, indem der Variable mit dem Gleichheitszeichen ein anderer Wert zugesprochen wird (passiert in diesem Programm nicht). Da sie aber bereits vom Typ her festgelegt ist, braucht „int“ dabei nicht erneut geschrieben werden.

- **Ändere** das Programm so **ab**, dass die grüne LED statt der gelben LED blinkt und dabei jeweils eineinhalb Sekunden an und eineinhalb Sekunden aus ist.

6. For-Schleifen

Das Blinken der LED basiert im Moment darauf, dass das Hauptprogramm immer wieder von vorne begonnen wird. Das stößt sehr schnell an seine Grenzen, z.B. bei diesem Problem: Die gelbe LED soll 10 Mal im Sekundenrhythmus (halbe an, halbe aus) blinken, anschließend soll die grüne LED für 5 Sekunden an sein. Das soll insgesamt ständig wiederholt werden.

Eine Lösung könnte so aussehen:

```
int LEDgelbPin =2;
int LEDgruenPin = 4;

void setup() {
  pinMode(LEDgelbPin, OUTPUT);
  pinMode(LEDgruenPin, OUTPUT);
}

void loop() {
  for(int i=0; i<10; i=i+1){
    digitalWrite(LEDgelbPin, HIGH);
    delay(500);
    digitalWrite(LEDgelbPin, LOW);
    delay(500);
  }
  digitalWrite(LEDgruenPin, HIGH);
  delay(5000);
  digitalWrite(LEDgruenPin, LOW);
}
```

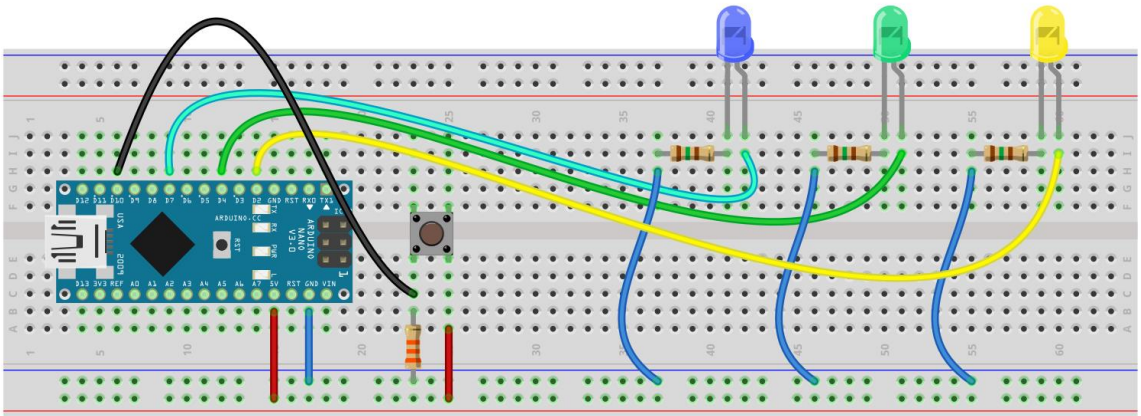
Dieser Sketch nutzt eine sogenannte **For-Schleife**. Diese erlaubt es, eine bestimmte Sequenz von Befehlen eine genau festgelegte Anzahl an Malen zu wiederholen – in diesem Fall 10 Mal.

Erklärung:

- `int i=0` erzeugt eine Zählvariable `i` mit dem Startwert 0.
- `i<10` gibt an, dass die Schleife so lange durchlaufen wird, wie `i` kleiner als 10 ist.
- `i=i+1` erhöht den Wert der Zählvariablen nach jedem Durchlauf um 1.
- Befehle die wiederholt werden → alles innerhalb der geschwungenen Klammern

In unserem Beispiel bedeutet das, dass der enthaltene Codeblock, nämlich das Ein- und Ausschalten der gelben LED mit jeweils 0,5 Sekunden Pause, genau 10 Mal durchlaufen wird. Danach wird der Schleifenbereich verlassen und der Code darunter wird ausgeführt.
Dort wird die grüne LED für 5 Sekunden angeschaltet (`delay(5000)`), bevor sie wieder ausgeschaltet wird. Da sich dieser gesamte Ablauf in der Funktion `loop()` befindet, wird er danach wieder von vorne begonnen; die LED-Sequenz wiederholt sich also endlos.

7. <u>Verzweigungen</u>



Manchmal reicht es nicht aus, dass ein Arduino-Programm einfach nur Befehle nacheinander ausführt oder Schleifen abarbeitet. Stattdessen soll das Verhalten des Programms **abhängig von**

bestimmten Bedingungen sein – zum Beispiel davon, ob ein Taster gedrückt wurde oder nicht. Solche Situationen erfordern eine **Verzweigung**.

Dazu verwendet man den Befehl `if`, der genau prüft, ob eine bestimmte Bedingung erfüllt ist. Falls ja, werden bestimmte Befehle ausgeführt. Ist die Bedingung **nicht** erfüllt, kann mit dem Zusatzbefehl `else` eine **alternative Anweisung** ausgeführt werden.

Diese Bedingungen können mit logischen Operatoren festgelegt werden. In Frage kommen z.B.:

- Gleichheit: `a==b`
- Ungleichheit: `a!=b`
- Größer bzw. Kleiner: `a>b` bzw `a<b`
- Größergleich bzw. Kleinergleich: `a>=b` bzw `a<=b`

Ein Beispiel siehst du im folgenden Sketch:

```
const int LEDgelbPin =2;
const int LEDgruenPin = 4;
const int TasterPin = 10;

int aktiv = 2;

void setup() {
  pinMode(LEDgelbPin, OUTPUT);
  pinMode(LEDgruenPin, OUTPUT);
  pinMode(TasterPin, INPUT);
}

void loop() {
  if(digitalRead(TasterPin)==HIGH){
    aktiv = LEDgelbPin;
  }
  else{
    aktiv = LEDgruenPin;
  }
  digitalWrite(aktiv, HIGH);
  delay(100);
  digitalWrite(aktiv, LOW);
}
```

Erklärung:

- Der Taster ist an Pin 10 angeschlossen und wird in der `setup()`-Funktion als Eingang definiert: `pinMode(TasterPin, INPUT);`.
- Im `loop` wird mit `digitalRead(TasterPin)` überprüft, ob der Taster gedrückt ist. Wenn ja (d.h. `digitalRead(...) == HIGH`), dann soll die **gelbe LED** blinken.
- Wenn der Taster **nicht** gedrückt ist (`else`), soll stattdessen die **grüne LED** blinken.

Die Variable `aktiv` wird hier verwendet um zu speichern, welcher LED-Pin aktuell „aktiv“ ist, also angesteuert werden soll.

Nach der Entscheidung (`if ... else`) wird genau dieser Pin für 100 Millisekunden eingeschaltet (`digitalWrite(aktiv, HIGH)`), dann wieder ausgeschaltet (`digitalWrite(aktiv, LOW)`). Das Ganze wiederholt sich ständig im loop.

Vielleicht ist dir aufgefallen, dass die Pins für die LEDs und den Taster in diesem Sketch mit `const int` statt wie bisher mit `int` deklariert wurden. Das hat den Hintergrund, dass die Pins im Laufe des Programms nicht geändert werden sollen, und daher als Konstanten gespeichert werden können.

- **Programmiere** einen Sketch, der das folgende Problem löst: Eine rote LED soll immer nicht leuchten, wenn ein Taster gerade gedrückt wird. In den Zeiten, in denen der Taster nicht gedrückt ist, soll sie dagegen eingeschaltet sein.

8. <u>While-Schleifen</u>

Manchmal möchte man, dass das Programm solange wartet, bis eine bestimmte Bedingung erfüllt ist, zum Beispiel bis ein Taster gedrückt wird. Für solche Fälle gibt es die sogenannte `while`-Schleife.

Sie funktioniert ähnlich wie die bereits bekannte `for`-Schleife, aber mit einem Unterschied: Die `while`-Schleife läuft so lange, wie eine bestimmte Bedingung wahr **(**`true`**)** ist; man weiß also vorher nicht unbedingt, wie oft die Schleife durchlaufen wird.

Ein Beispiel siehst du im folgenden Sketch:

```
const int LEDgelbPin =2;
const int TasterPin = 10;

void setup() {
  pinMode(LEDgelbPin, OUTPUT);
  pinMode(TasterPin, INPUT);
}

void loop() {
  while(digitalRead(TasterPin)==LOW){
    delay(100);
  }
  digitalWrite(LEDgelbPin, HIGH);
  delay(1000);
  digitalWrite(LEDgelbPin, LOW);
}
```

Erklärung:

- Der Sketch wartet darauf, dass der Taster (an Pin 10) gedrückt wird.
- Mit `digitalRead(TasterPin) == LOW` wird geprüft, ob der Taster nicht gedrückt ist (das ist der Normalzustand bei Tastern mit Pull-Down-Widerstand, d.h. dass der Widerstand den Pin bei nicht gedrücktem Taster auf ground zieht).
- Solange der Taster nicht gedrückt ist, bleibt das Programm in der `while`-Schleife und macht alle 100 Millisekunden eine kurze Pause (`delay(100)`). So wird unnötiges „Rasen“ durch die Schleife vermieden.
- Erst wenn der Taster gedrückt wird, die Bedingung also nicht mehr erfüllt ist, wird die Schleife verlassen, die LED an Pin 2 eingeschaltet und für eine Sekunde gewartet (`delay(1000)`), bevor sie wieder ausgeschaltet wird.

9. Unterprogramme

Schauen wir uns das folgende Programm mal genauer an:

```
const int LEDgelbPin = 2;
const int LEDgruenPin = 4;
const int TasterPin = 10;

void setup() {
  pinMode(LEDgelbPin, OUTPUT);
  pinMode(LEDgruenPin, OUTPUT);
  pinMode(TasterPin, INPUT);
}

void loop() {
  while(digitalRead(TasterPin)==LOW){
    delay(100);
  }
  for(int i=0; i<3; i=i+1){
    digitalWrite(LEDgelbPin, HIGH);
    delay(500);
    digitalWrite(LEDgelbPin, LOW);
    delay(500);
  }
  for(int i=0; i<7; i=i+1){
    digitalWrite(LEDgruenPin, HIGH);
    delay(500);
    digitalWrite(LEDgruenPin, LOW);
    delay(500);
  }
}
```

Das Programm ist nicht schwierig zu verstehen: Es wird auf den Druck eines Tasters gewartet. Wenn das passiert, dann leuchtet zunächst eine gelbe LED drei Mal auf, anschließend eine grüne LED sieben Mal.
Was dabei auffällt: Die beiden Schleifen für die LED-Ausgaben sind sehr ähnlich und unterscheiden sich eigentlich nur darin, welche LED angesprochen wird und wie oft sie aufleuchten soll. Es wäre also sehr praktisch, wenn es eine fertige Routine für das Blinken einer LED gäbe, der man nur noch mitteilen müsste, wie oft welche LED blinken soll.

Diesen Wunsch erfüllen „Unterprogramme“, wie das folgende Beispiel demonstriert:

```
const int LEDgelbPin = 2;
const int LEDgruenPin = 4;
const int TasterPin = 10;

void setup() {
  pinMode(LEDgelbPin, OUTPUT);
  pinMode(LEDgruenPin, OUTPUT);
  pinMode(TasterPin, INPUT);

}

void loop() {
  while(digitalRead(TasterPin)==LOW){
    delay(100);
  }
  blinken(LEDgelbPin,3);
  blinken(LEDgruenPin,7);
}

void blinken(int PinNr, int anzahl){
  for(int i=0; i<anzahl; i=i+1){
    digitalWrite(PinNr, HIGH);
    delay(500);
    digitalWrite(PinNr, LOW);
    delay(500);
  }
}
```

Neben `setup` und `loop` tritt hier in ganz ähnlicher Struktur das Unterprogramm `blinken` auf. Mit der Zeile `void blinken(int PinNr, int anzahl)` wird

- zunächst der frei wählbare Name des Unterprogramms festgelegt (hier `blinken`),
- es wird ebenfalls angelegt, dass diesem Unterprogramm zwei Integerwerte übergeben werden, die hier in der Pinnummer der LED und der Anzahl der vorgesehenen Blinkvorgänge bestehen,
- und es wird mit dem vorgestellten `void` ausgedrückt, dass dieses Unterprogramm nach dem Abarbeiten keine Werte an das aufrufende Programm zurückgibt.

Im anschließenden Programmblock in den geschweiften Klammern folgt der Programmcode des Unterprogramms. Wahrscheinlich erkennst du die Schleifenstruktur wieder, die im Ausgangsprogramm zweimal vorkam. Im Schleifenkopf in der Zählbedingung steht jetzt lediglich statt einer konkreten Zahl von Blinkvorgängen die Variable `anzahl` und im `digitalWrite`-Befehl statt eines konkreten Pins die Variable `PinNr`.

Daher arbeitet das Unterprogramm ganz abhängig davon, wie es aufgerufen wird. Der erste Aufruf erfolgt aus dem `loop` heraus mit der Befehlszeile `blinken(LEDgelbPin,3);` .Hier wird also die Schleife dreimal ausgeführt und jeweils die gelbe LED ein- und ausgeschaltet. Bei der darauffolgenden Zeile `blinken(LEDgruenPin,7);` wird dagegen für ein siebenmaliges Blinken der grünen LED gesorgt.

Allgemein erfolgt der Aufruf eines Unterprogramms durch Nennung seines Namens (ohne vorangestelltes `void`) und der Übergabe der kommagetrennten Variablenwerte in runden Klammern.

- **Ändere** das Unterprogramm **so ab, dass** es als weitere Variablen die Leuchtdauer der LED und die anschließende Zeit in ausgeschaltetem Zustand übergeben bekommt und im Ablauf berücksichtigt. Die neuen Aufrufzeilen aus dem loop heraus sollen jetzt lauten:
  `blinken(LEDgelbPin, 3, 700, 300);` und
  `blinken(LEDgruenPin, 7, 250, 750);`

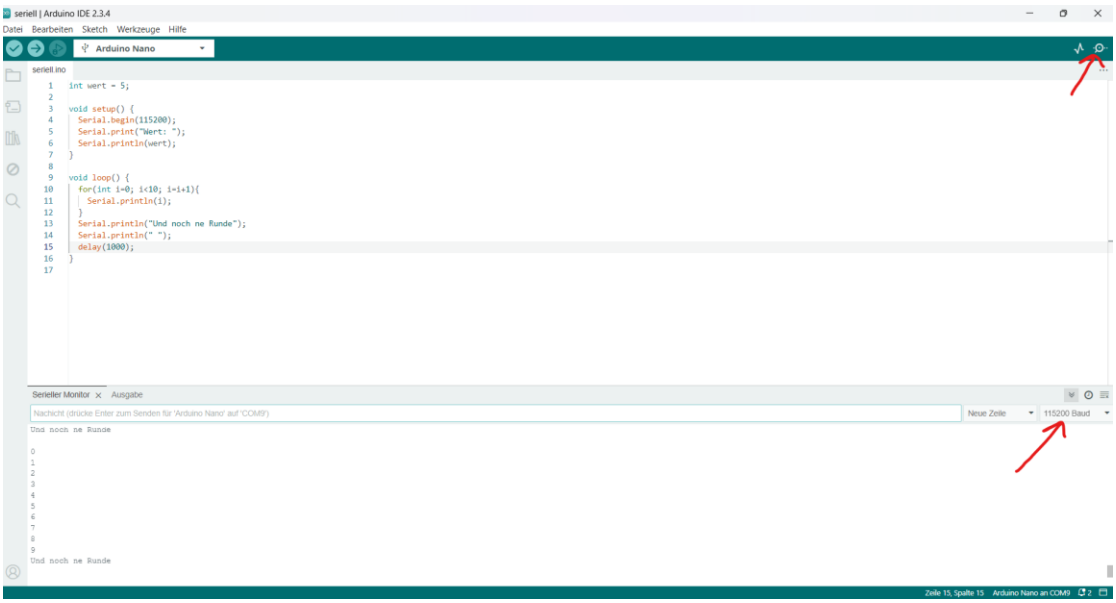
10. Der serielle Monitor und String-Variablen

In der Grundausstattung hat der Arduino zwar kein Display, er kann aber über die USB-Schnittstelle und den angeschlossenen Computer bzw. seinen Monitor Schrift, Sensorwerte, … ausgeben. Dazu muss im setup-Bereich die serielle Kommunikation mit dem Befehl `Serial.begin` begonnen werden und es muss dahinter in Klammern eine Übertragungsrate („Baudrate“) festgelegt werden. Anschließend können mit dem Befehl `Serial.print` Text, Werte, … ausgegeben werden. Wird sogar `Serial.println` (`ln` steht für „line“) verwendet, dann wird zusätzlich am Ende der Ausgabe ein Zeilenumbruch eingefügt.

- **Öffne** einen neuen Sketch **und probiere** ein wenig mit Änderungen an dem folgenden kurzen Sketch herum:

```
1   int wert = 5;
2
3   void setup() {
4     Serial.begin(115200);
5     Serial.print("Wert: ");
6     Serial.println(wert);
7   }
8
9   void loop() {
10    for(int i=0; i<10; i=i+1){
11      Serial.println(i);
12    }
13    Serial.println("Und noch ne Runde");
14    Serial.println(" ");
15  }
```

Damit die Ausgabe auf dem Monitor erscheint, muss der serielle Monitor in der Arduino-IDE geöffnet werden, was mit dem Symbol oben rechts in der Ecke passiert. Außerdem muss die richtige Baudrate im seriellen Monitor gewählt werden.



Ein Blick auf die Zeilen mit den Ausgabebefehlen im `setup` zeigt zwei Varianten:

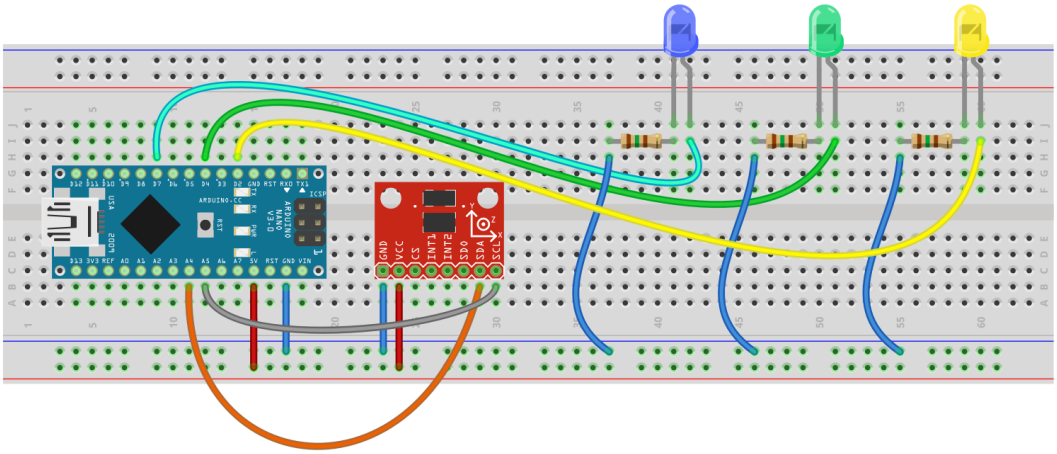
```
Serial.print("Wert: ");
Serial.println(wert);
```

In der oberen Zeile steht im Befehl ein Text, der genau in dieser Form im seriellen Monitor ausgegeben werden soll. Dafür muss er in Anführungszeichen gesetzt werden und man spricht von einem sogenannten „String“. In der unteren Zeile wird dagegen der Zahlenwert ausgegeben, der in der Integer-Variable „wert“ hinterlegt ist.
Es ist auch möglich, Strings in Variablen zu hinterlegen. Dafür muss die Variable benannt werden und dem Typ „String“ zugeordnet werden:

```
int wert = 5;
String bezeichnung = "Wert: ";

void setup() {
 Serial.begin(115200);
 Serial.print(bezeichnung);
 Serial.println(wert);
}
```

11. Den Beschleunigungssensor ADXL345 einbinden

- **Verändere** die Schaltung auf dem Breadboard so, dass sie dem folgenden Aufbau entspricht:

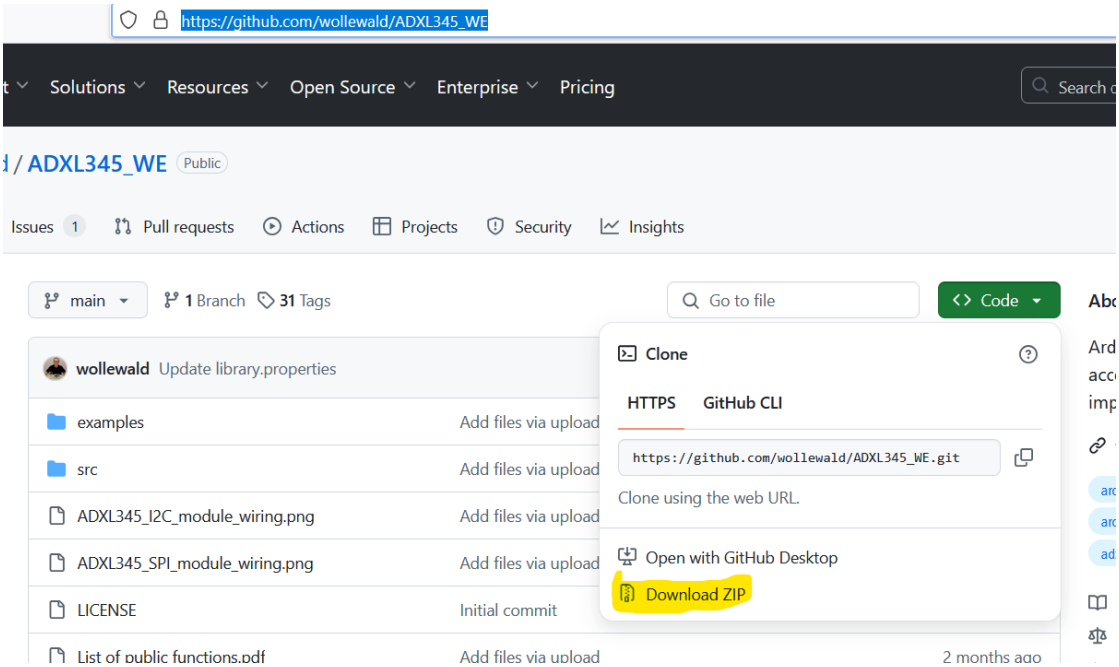


Für den Sensor ADXL345, mit dem die Würfelausrichtung ausgelesen werden soll, nutzen wir keine digitalen Pins. Stattdessen wird der serielle Datenbus „I2C“ eingesetzt, bei dem die Kommunikation in Form von Datenpaketen stattfindet, die nacheinander über eine Datenleitung versendet werden. Damit die Datenpakete richtig verstanden werden, muss dies über eine weitere Leitung, die sogenannte „Clockline“, synchronisiert werden. Der Mikrocontroller agiert und spielt die Rolle des „Masters“. Der Sensor reagiert und ist in der Rolle des sogenannten „Slaves“. Es wäre sogar möglich, bis zu 127 Slaves an einem Datenbus zu betreiben, aber es kann nur einen Master pro Bus geben. Daher erhält jeder Slave eine eigene Adresse, unter der er angesprochen werden kann.

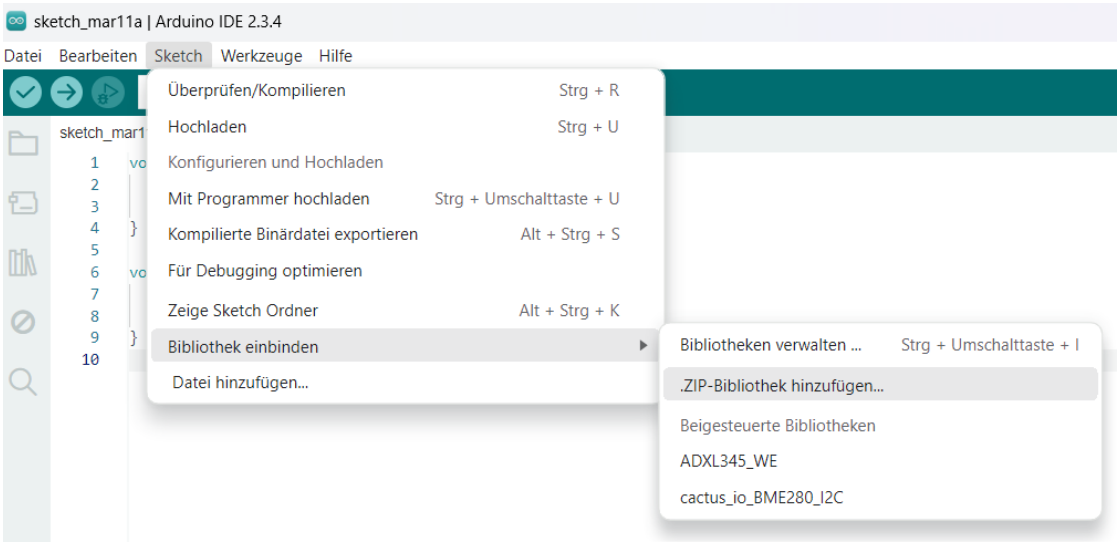
Da es sehr aufwändig ist, für einen Sensor dieser Art (oder andere Komponenten) alles selbst zu programmieren, gibt es in vielen Fällen sogenannte Bibliotheken, die die Arbeit deutlich vereinfachen. Sie enthalten fertig programmierten Hilfscode, der in eigene Programme eingefügt werden kann und dann Funktionen für den Sensor enthält, auf die mit entsprechenden Befehlen zurückgegriffen werden kann. Oft stammen die Bibliotheken von den Herstellern der Bauteile, aber es gibt auch eine große Arduino-Community, die solche Hilfen frei zur Verfügung stellen. Bevor du eine Bibliothek installierst, ist es sinnvoll unter „Sketch“→ „Bibliothek einbinden“ nachzusehen, ob die gewünschte Bibliothek bereits unter „Beigesteuerte Bibliotheken“ aufgeführt ist und damit schon installiert ist.

Wir nutzen eine Bibliothek von Wolfgang Ewald, die dieser sehr ausführlich auf seiner eigenen Website vorstellt: https://wolles-elektronikkiste.de/adxl345-teil-1 bzw.
https://wolles-elektronikkiste.de/adxl345-der-universelle-beschleunigungssensor-teil-2

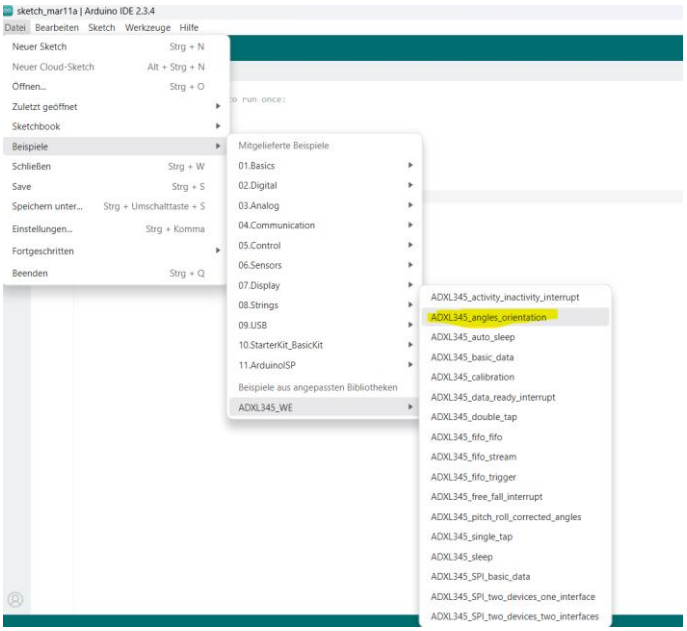
Für den Download muss man die folgende Seite aufsuchen:
https://github.com/wollewald/ADXL345_WE



Ist das ZIP-Verzeichnis heruntergeladen, dann kann es in der Arduino-IDE eingebunden werden. Dazu kann man unter „Sketch“ den Unterpunkt „Bibliothek einbinden“ wählen und die ZIP-Datei einfügen, ohne dass sie vorher extrahiert werden muss:



Für uns ist der Beispielsketch Nr. 3 der Website interessant, da er unter anderem die Orientierung des Sensors ausliest. Ist die Bibliothek eingebunden, dann steht er unter den mitgelieferten Beispielen zur Verfügung:



12. <u>Werte des ADXL345 auslesen</u>

- **Öffne** einen neuen Sketch und das eben genannte Beispielprogramm. **Kopiere dann** zunächst den gesamten Beispielcode in den neuen Sketch. **Lösche anschließend** soweit alles, dass nur noch die folgenden Zeilen übrigbleiben (, die hier noch zusätzlich kommentiert sind):

```
#include<Wire.h>
#include<ADXL345_WE.h>            // hier iwrd die Bibliothek für den Sensor eingebunden
#define ADXL345_I2CADDR 0x53      // hier wird die Sensoradresse festgelegt

ADXL345_WE myAcc = ADXL345_WE(ADXL345_I2CADDR); //ab sofort kann der Sensor mit "myAcc" angesprochen werden

void setup(){
  Wire.begin();
  Serial.begin(115200);              // die serielle Kommunikation wird mit der Übertragungsrate 115200 gestartet
  Serial.println("ADXL345_Sketch");
  Serial.println();

  if(!myAcc.init()){                          // falls der Sensor nicht verbunden ist
    Serial.println("ADXL345 not connected!");
  }

  Serial.println("Position your ADXL345 flat and don't move it");   //es folgen einige Kalibrierungsvorgänge und Festlegungen
  delay(2000);
  myAcc.measureAngleOffsets();
  Serial.println("....done");

  myAcc.setDataRate(ADXL345_DATA_RATE_50);
  Serial.print("Data rate: ");
  Serial.print(myAcc.getDataRateAsString());

  myAcc.setRange(ADXL345_RANGE_2G);
  Serial.print("  /  g-Range: ");
  Serial.println(myAcc.getRangeAsString());
  Serial.println();
}

void loop() {
   Serial.print("Orientation of the module: ");
  Serial.println(myAcc.getOrientationAsString());   //Die Orientierung wird ausgelesen und ausgegeben

  Serial.println();

  delay(1000);   //eine Sekunde warten vor Neubeginn der Schleife
}
```

- **Probiere** das Programm mal **aus, spiele** etwas mit der Orientierung des Sensors **und schau dir dabei** die Werte im seriellen Monitor **an**.

13. Nächste Schritte

Die Grundlagen für das Projekt sind damit gelegt. Jetzt folgen drei weitere Schritte:

i. In Teil 2 wird das eigentliche Programm für den Escapewürfel erarbeitet.
ii. Die elektrische Schaltung sollte aus dem vorläufigen Aufbau auf dem Breadboard in eine dauerhafte gelötete Form auf einer Platine überführt werden.
iii. Der Würfel muss zusammengebaut und gestaltet werden.

Zum Abschluss ein Blick auf eine mögliche Anordnung und Verkabelung der Bauteile auf einer Lochrasterplatine: