

# VLC Workshop

Workshop zur Kommunikation mit sichtbarem Licht am Fachgebiet Lichttechnik



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



## 1 Einführung

Für diesen Workshop werden wir einen Arduino Nano mit dem FGLT-VLC-Shield programmieren, um mit sichtbarem Licht zwischen zwei Einheiten zu kommunizieren (Visible Light Communication - VLC). Dazu werden wir die Arduino IDE nutzen. Sie ermöglicht eine einfache Programmierung von unterstützten Microcontrollern.

### 1.1 Die Arduino Programmierumgebung

Um mit der Programmierung unserer Microcontroller anfangen zu können müssen wir zuerst die Entwicklungs-umgebung starten. Diese befindet sich direkt auf dem Desktop der Laptops oder kann Zuhause unter <https://www.arduino.cc/en/Main/Software> kostenfrei runtergeladen werden. Wenn ihr die Umgebung gestartet habt müsst ihr noch sicherstellen, dass das richtige Board ausgewählt ist. Dieses findet ihr in **Werkzeuge > Board > Arduino Nano**. Unsere Nanos haben noch den alten sogenannten "Bootloader". Deshalb müsst ihr das noch unter **Werkzeuge > Prozessor > ATmega328P (Old Bootloader)** einstellen.

### 1.2 Blink

Wir wollen langsam anfangen. Das einfachste, was man auf einem Mikrocontroller machen kann, ist eine LED blinken zu lassen. Hierzu schauen wir uns erst den Text, der bereits in der Programmierumgebung steht genauer an: `void setup()` und `void loop()`.

Diese sind Funktionen, die zur Programmierung eines Arduinos nötig sind. Sie haben die folgenden Zwecke:

- `void setup()` - Diese Funktion wird sobald der Arduino Strom bekommt als Initialisierung **einmal** zu Beginn des Programms durchlaufen.
- `void loop()` - Diese Funktion wird nach dem Setup zur Laufzeit des Programmes **ständig** in einer Endlos-schleife aufgerufen.

In diese Funktionen - besser gesagt zwischen die geschweiften Klammern dieser Funktionen - soll nun euer Programm geschrieben werden. In der Regel wird in jede Zeile nur ein Befehl geschrieben und dann mit einem Semikolon (";") abgeschlossen. Die Zeilen werden nacheinander von oben nach unten ausgeführt. Das geschieht sehr schnell.

Dem Arduino muss also irgendwie gesagt werden, dass er seine LED ein- und dann wieder ausschalten soll. Ein- und ausschalten heißt hier folgendes: Es liegt eine Spannung an der LED an (bzw. Strom ist an) oder es liegt keine Spannung an (Strom sitzt aus), ähnlich wie bei einem Schalter. Wollt ihr diesen Schalter im Arduino bedienen, sprecht ihr eigentlich einen Pin an. Um diesem Pin zu sagen, er soll eine Spannung ausgeben, müsst ihr dem Arduino erst sagen, dass der Pin ein Ausgang sein soll. Danach könnt ihr ihm sagen, er soll den Pin, bzw. die LED ein- und ausschalten.

Die Reihenfolge ist also:

1. Dem Arduino sagen, dass die LED ein Ausgang sein soll
2. Dem Arduino sagen, er soll die LED einschalten
3. Dem Arduino sagen, er soll die LED ausschalten
4. Dem Arduino sagen, er soll Schritte 2 und 3 immer wiederholen

Schritt 1 können wir mit der Funktion bzw. Befehl `pinMode(Pin, Modus);` erreichen. Sie nimmt zwei Parameter: Der Pin, der konfiguriert werden soll und der Modus, in dem der Pin sein soll. Hier soll der Pin der LED als Ausgang konfiguriert werden. Der Pin hat den Namen `LED_BUILTIN`. Der Modus für "Ausgang" ist `OUTPUT`. Da diese Funktion nur einmal ausgeführt werden muss, sollte sie in die `setup()` Funktion geschrieben werden. Um beim Programmieren dem Programm zu sagen, dass eine Anweisung fertig ist, wird ein Semikolon (`;`) genutzt.

Zwischen die geschweiften Klammern von `setup()` müsst ihr also folgendes schreiben:

```
pinMode(LED_BUILTIN, OUTPUT);
```

Es gibt im Übrigen auch noch einen Modus für "Eingang": `INPUT`. Diesen werdet ihr später nochmal brauchen.

Für Schritte 2 und 3 braucht ihr die gleiche Funktion: `digitalWrite(Pin, Ausgabewert)`. Mit ihr könnt ihr auf einem als Ausgang konfigurierten Pin eine hohe oder niedrige Spannung anlegen, ihn also an- oder ausschalten. Die Ausgabewerte werden mit `HIGH` und `LOW` bezeichnet. Um die LED anzuschalten kann also folgende Anweisung genutzt werden:

```
digitalWrite(LED_BUILTIN, HIGH);
```

Analog dazu müsst ihr einfach `HIGH` mit `LOW` ersetzen, um sie wieder auszuschalten.

Da diese beiden Funktionen immer wieder nacheinander ausgeführt werden sollen, kommen sie in die `loop()`-Funktion. Damit sollte das Programm in etwa so aussehen:

```

1 // die setup Funktion wird am Anfang einmal ausgeführt
2 void setup() {
3 // die blaue LED auf Pin 5 wird als Ausgang konfiguriert
4 pinMode(LED_BUILTIN, OUTPUT);
5 }
6
7 // die loop Funktion wird wieder und wieder für immer ausgeführt
8 void loop() {
9 digitalWrite(LED_BUILTIN, HIGH);      // schalte die LED ein (setze den Pegel der LED auf HIGH)
10 digitalWrite(LED_BUILTIN, LOW);       // schalte die LED aus (setzte den Pegel der LED auf LOW)
11 }

```

Der Text hinter den doppelten Schrägstrichen (//) wird als Kommentar gewertet und vom Arduino ignoriert.

### 1.3 Hochladen

Um nun die LED des Arduino blinken zu lassen müsst ihr das geschriebene auf das Board laden. Hierzu müsst ihr es an den Laptop anschließen und gegebenenfalls noch unter **Werkzeuge > Port** auswählen. Anschließend könnt ihr es über die runde Schaltfläche oben links, in der ein Pfeil nach rechts zu sehen ist hochladen. Alternativ könnt ihr das auch in **Sketch > Hochladen** machen. Sollten beim Kompilieren oder Übertragen irgendwelche Fehler aufgetreten sein könnt ihr sie im unteren Ausgabefenster sehen.

Nun sollte die LED auf dem Board leuchten.

### 1.4 Warten

Warum leuchtet aber nun die LED nur, statt zu blinken?

Das liegt einfach daran, dass sie so schnell ein- und wieder ausgeschaltet wird, dass wir diese Wechsel gar nicht mehr wahrnehmen können. Um die Übergänge sehen zu können, muss dem Arduino gesagt werden, dass er zwischen dem Ein- und Ausschalten warten soll. Auch hierzu gibt es wieder eine Funktion: `delay(Millisekunden)`. Wenn sie aufgerufen wird, wartet der Arduino für die angegebene Anzahl an Millisekunden - also tausendst Sekunden.

Wir wollen nun den Arduino zwischen dem Umschalten je eine Sekunde warten lassen. Da eine Sekunde aus 1000 Millisekunden besteht, geht dies mit der Anweisung

```
delay(1000);
```

Diese Funktion muss sowohl nach dem Aus- als auch nach dem Einschalten aufgerufen werden. Das fertige Programm müsste also nun wie folgt aussehen:

```

1 // die setup Funktion wird am Anfang einmal ausgeführt
2 void setup() {
3 // die blaue LED auf Pin 5 wird als Ausgang konfiguriert
4 pinMode(LED_BUILTIN, OUTPUT);
5 }
6
7 // die loop Funktion wird wieder und wieder für immer ausgeführt
8 void loop() {
9 digitalWrite(LED_BUILTIN, HIGH);      // schalte die LED ein (setze den Pegel der LED auf HIGH)
10 delay(1000);                      // warte 1000ms (eine Sekunde)
11 digitalWrite(LED_BUILTIN, LOW);       // schalte die LED aus (setze den Pegel der LED auf LOW)
12 delay(1000);                      // warte wieder 1000ms
13 }

```

Im Anhang dieser Anleitung könnt ihr eine kleine Referenz zur Programmierung und zu den für diesen Workshop wichtigsten Funktionen finden.

### 1.5 Beispielprogramme

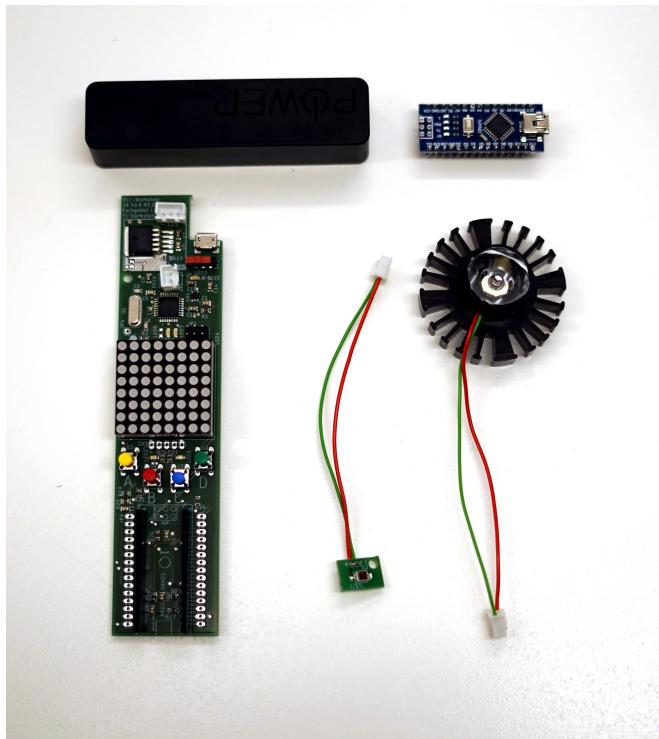
Über die Arduino IDE könnt ihr auch Beispielprogramme öffnen und verwenden. Ein solches Beispiel heißt "Blink". Es macht genau das was wir in der vorgehenden Aufgabe gemacht haben: Eine LED auf dem Arduino Nano zum blinken bringen. Das Beispiel findet ihr unter **Datei > Beispiele > 01.Basics > Blink**.

---

## 2 Komponenten des VLC Microcontrollers

---

Für diesen Workshop haben wir den Arduino etwas erweitert, um mit ihm die Kommunikation mit sichtbarem Licht zu ermöglichen. In Abbildung 1 könnt ihr die Bestandteile sehen:



**Abbildung 1:** Die Komponenten des VLC Microcontrollers (ohne Gehäuse)

Zu sehen sind:

1. Oben links die Powerbank
2. Oben rechts der Arduino Nano
3. Unten links das VLC Shield
4. In der Mitte die Fotodiode
5. Ganz rechts die LED mit Kühlkörper

Auf dem Shield selbst befinden sich vier Taster (Tasten), zwei LEDs, ein LED Display und ein An- und Ausschalter. Der Arduino steckt bereits auf dem Shield.

---

## 3 Zusammenbau des VLC Microcontrollers

---

Den Controller baut ihr wie folgt zusammen:

Die Powerbank kommt unten in das Gehäuse. Darauf kommt das VLC Shield. Danach könnt ihr die LED und Fotodiode vorne im Gehäuse an die vorgesehenen Stellen platzieren. Anschließend werden die Fotodiode und die LED an das Shield angeschlossen, wie in Abbildung 2 zu sehen.



**Abbildung 2:** Das VLC Shield mit den Anschlüssen für die Fotodiode und die LED markiert.

#### 4 Pinbelegungen

2	LED Matrix
3	Fotodiode mit Filter
4	LED mit Treiber
5	Blaue Status-LED
6	Rote Status-LED
7	Gelber Taster
8	Roter Taster
9	Blauer Taster
A1	Grüner Taster

## 5 Einstiegsaufgabe - Blink 2

### 5.1 Blink mit dem VLC Shield

Nun stecken wir den Mikrocontroller auf das VLC Shield. Wie bereits erwähnt, befinden sich auf dem Shield auch zwei LEDs, die angesteuert werden können: Eine Blaue auf Pin 5 und eine Rote auf Pin 6. Wir wollen jetzt das Blink Programm für die blaue LED umschreiben. Hierzu muss nur überall im bestehenden Programm `LED_BUILTIN` durch 5 ersetzt werden. Dadurch wird nun nicht mehr die definierte LED, sondern die blaue LED an Pin 5 angesteuert.

### 5.2 Blink mit zwei LEDs

Als nächstes soll jetzt auch noch die rote LED auf dem Board abwechselnd mit der Blauen blinken. Konfiguriert sie hierzu als Ausgang und schaltet sie immer dann an, wenn ihr die blaue LED ausschaltet und umgekehrt.

## 6 Taster

Wir wollen nun ein weiteres externes Element in unsere Steuerung aufnehmen: die Taster. Auf unserem Board befinden sich unter dem Display vier davon: Gelb, Rot, Blau und Grün.

In diesem Schritt ist eure Aufgabe, die blaue LED leuchten zu lassen, wenn der blaue Taster gedrückt ist. Der blaue Taster ist an Pin 9 angeschlossen.

### 6.1 Eingänge

Um einen Pin als Eingang zu verwenden, muss er zunächst als Eingang gesetzt werden. Dazu wird wieder `pinMode` verwendet, aber mit dem Unterschied, dass der Pin dieses Mal kein Ausgang sein soll.

Um später den aktuellen Zustand eines Pins zu lesen, wird `digitalRead(Pin)` verwendet. Wenn der Pin gerade mit "Ground"(bzw. LOW) verbunden ist liefert diese Funktion 0, bzw. logisch falsch zurück. Ist er gerade mit der Spannungsversorgung (bzw. HIGH) verbunden, liefert sie 1, bzw. logisch wahr zurück. Ist der Pin überhaupt nicht verbunden, dann liefert `digitalRead` ziemlich zufällig 0 oder 1. Da sich mit zufälligen Werten schlecht rechnen lässt, sollte man darauf achten, dass das nie passiert. Das kann allerdings auf unserem Board nicht geschehen, da alle Taster bereits verbunden sind.

### 6.2 Die Implementierung

Um das oben beschriebene Programm zu verwirklichen müssen folgende Schritte befolgt werden:

1. Den blauen Taster als **Eingang** (INPUT) konfigurieren
2. Die blaue LED als **Ausgang** (OUTPUT) konfigurieren
3. Den Arduino fragen, ob der Taster gerade gedrückt wird
  - 3.1. Falls der Taster gedrückt ist, soll die blaue LED eingeschaltet werden
  - 3.2. Falls der Taster nicht gedrückt ist, soll die blaue LED ausgeschaltet werden
4. Schritt 3 soll immer wieder ausgeführt werden

#### 6.2.1 Fallunterscheidungen

Eine Fallunterscheidung kann beim Programmieren mit einer sogenannten **if**-Anweisung erreicht werden. Eine solche Anweisung besteht aus den folgenden Teilen:

- Eine Bedingung, die entweder wahr und falsch sein kann
- Ein **if**-Pfad, der ausgeführt wird, wenn die genannte Bedingung zutrifft
- Ein **else**-Pfad, der ausgeführt wird, wenn die Bedingung nicht erfüllt ist. (optional)

Im Programm sieht das dann aus wie folgt:

```
1 if(Bedingung) {  
2     // Code, der ausgeführt werden soll, wenn die Bedingung erfüllt ist  
3 } else {  
4     // Code, der andernfalls ausgeführt werden soll.  
5 }
```

## 6.2.2 Vergleiche

Beim Programmieren gibt es verschiedene Vergleiche, die man machen kann. Viele davon kommen aus der Mathematik. Im folgenden seht ihr einige Beispiele, wie man zwei Ausdrücke (A und B) vergleichen kann:

- Gleichheit: Um zu prüfen, ob A und B gleich sind kann == genutzt werden: `A == B`
- Kleiner als: Kann genau so aufgeschrieben werden: `A < B`
- Größer als: Kann auch genau so aufgeschrieben werden: `A > B`
- Kleiner-gleich/größer-gleich: Kann auch so aufgeschrieben werden: `A <= B` oder `A >= B`

Man kann auch verschiedene Vergleiche oder Bedingungen miteinander verknüpfen:

- Wenn Bedingung X und Y zutreffen sollen, kann `&&` (und) genutzt werden: `x && y`
- Wenn eine der beiden Bedingungen zutreffen soll wird `||` (oder) verwendet: `x || y`

Kommen wir nun zum Taster zurück: Wenn der Taster gedrückt ist, wird `digitalRead HIGH` zurückgeben. Die entsprechende Bedingung hierzu ist `digitalRead(Pin)== HIGH`. Hierbei muss "Pin" noch mit dem Pin des blauen Tasters ersetzt werden.

Das Programm hat damit folgende Struktur

```
1 void setup() {  
2     //hier den Taster als Eingang und die LED als Ausgang konfigurieren  
3 }  
4  
5 void loop() {  
6     if( /*Bedingung*/ ) { // hier die Bedingung "Ist der Taster gedrückt" abgefragt  
7         // LED einschalten  
8     } else {  
9         // LED ausschalten  
10    }  
11 }
```

Versucht, das Programm zu vervollständigen.

## 7 Taster 2

Nachdem ihr jetzt eine LED mit einem Taster leuchten lassen könnt soll das Ganze gleichzeitig auch noch mit dem roten Taster (an Pin 8) und der roten LED gemacht werden. Dazu muss an sich nur das Gleiche nochmal gemacht werden: Die neue LED und den neuen Taster konfigurieren, die neue Bedingung aufstellen, und die LED je nach dem Ergebnis ein- oder ausschalten.

An dieser Stelle wollen wir nun eine weiteres Element der Programmierung einführen: Variablen.

## 7.1 Variablen

Ähnlich wie in der Mathematik können auch beim Programmieren Variablen genutzt werden. Sie speichern Daten, die sie zugewiesen bekommen und können nach ihren Werten gefragt werden. Eine Variable hat immer einen Namen und einen Datentyp. Der Name wird genutzt, um sie aufrufen zu können, während der Datentyp festlegt, welche Art von Werten die Variable speichern kann.

Bevor eine Variable genutzt werden kann, muss sie zuerst angelegt werden. Hierzu wird folgende Syntax genutzt:  
`Datentyp Name;` Es gibt unter vielen anderen folgende Datentypen, auf die wir uns vorerst fokussieren wollen:

- `int`: Hält eine ganze, positive oder auch negative Zahl
- `String`: Hält einen Text
- `boolean`: Hält einen Wahrheitswert, also wahr (`true`) oder falsch (`false`)

Im Laufe des Workshops werden wir noch weitere Datentypen kennenlernen und nutzen.

Um zum Beispiel den Pin der blauen LED (eine ganze Zahl) zu speichern, kann folgender Ausdruck genutzt werden:

```
int blaueLED = 5;
```

Der Datentyp darf und muss hierbei nur beim ersten Anlegen angegeben werden. Solltet ihr den Wert dieser Variable also später ändern wollen müsst ihr also stattdessen folgenden Ausdruck nutzen:

```
blaueLED = 6;
```

Um den Wert der Variable auszulesen müsst ihr sie lediglich hinschreiben. So wird zum Beispiel `pinMode(5, OUTPUT);` zu `pinMode(blaueLED, OUTPUT);`. Dies trägt zur Lesbarkeit und zur einfacheren Änderung des Codes bei. Nichts anderes habt ihr übrigens mit `LED_BUILTIN, INPUT,` und den anderen Ausdrücken gemacht.

Nutzt bitte im oben beschriebenen Programm (Taster 2) Variablen, um die verschiedenen Pins zu speichern.

**HINWEIS:** Wenn ihr eine Variable innerhalb einer Funktion anlegt (wie zum Beispiel `setup()` oder `loop()`), könnt ihr sie nur innerhalb dieser Funktion nutzen! Wenn ihr eine Variable also in beiden Funktionen nutzen wollt, müsst ihr sie vor diesen beiden Funktionen anlegen.

## 8 Serielle Kommunikation

In diesem Workshop werden wir immer wieder serielle Kommunikation nutzen, um Nachrichten vom Arduino auf dem Laptop anzeigen zu lassen.

Zuerst müsst ihr in der `setup` Funktion die serielle Kommunikation initialisieren. Dies machen wir mit Hilfe des Befehls `Serial.begin(9600)`. Er startet auf dem Arduino eine serielle Kommunikation mit einer sogenannten Baud-Rate von 9600. Das bedeutet, dass er über seine serielle Schnittstelle 9600 Zeichen pro Sekunde senden wird. Mit `Serial.print(Nachricht)` und `Serial.println(Nachricht)` könnt ihr nun Strings (also Text) senden. Der Text muss in Anführungszeichen an die Funktion übergeben werden. Nach `print` wird in der gleichen Zeile weiter geschrieben, während `println` die Zeile danach beendet.

Um gesendeten Text lesen zu können müsst ihr den seriellen Monitor öffnen. Diesen könnt ihr entweder oben rechts in der Programmierumgebung oder unter  `Werkzeuge > Serieller Monitor` finden.

Versucht nun in einem kleinen Programm eine kurze Nachricht einmalig in der `setup` Funktion zu senden.

Danach könnt ihr versuchen, eine weitere Nachricht in der `loop` Funktion zu senden um zu sehen was passiert.

Es kann übrigens nicht nur Text, sondern auch Werte gesendet werden.

## 9 LED

Wir wollen nun einen kurzen Blick auf Frequenzen in Verbindung mit Licht werfen, bevor wir uns weiter mit den restlichen Komponenten der VLC-Shields beschäftigen.

### 9.1 Frequenzen

Eine Frequenz beschreibt eine Ereignisrate. Beispielsweise sind die BPM (Beats per minute), also das Tempo in der Musik eine Frequenzangabe. Eine Frequenz von 60 BPM entspricht 60 Schlägen pro Minute, also einem Schlag pro Sekunde. Somit hat das Tempo bei 60 BPM die Frequenz 1Hz (Hertz). Hierbei ist 1Hz = 1 Ereignis pro Sekunde.

Als Periode wird die Zeit zwischen zwei Ereignissen beschrieben. Bei einer Frequenz von 1Hz gäbe es also eine Periode von 1s. Die Umrechnung hier ist relativ simpel:

$$\text{Frequenz } f = \frac{1}{T} = \frac{1}{\text{Periode}}$$
$$\text{Periode } T = \frac{1}{f} = \frac{1}{\text{Frequenz}}$$

Bei einer Frequenz von 50Hz liegt also eine Periode von 0,02s oder 20ms vor.

Haben wir eine Periode von 5ms, so ist die Frequenz 200Hz.

Die meisten Lampen oder LEDs flimmern, werden also mehrmals die Sekunde ein- und wieder ausgeschaltet. Genau wie im ersten Programm (Blink) ohne das Warten sehen wir das allerdings nicht. Die Frage stellt sich nun, ab welcher Frequenz wir das flimmern nicht mehr wahrnehmen.

### 9.2 Flimmer

Wir wollen nun ein Programm schreiben, das sich sehr ähnlich wie "Blink" verhält. Am Anfang des Programms wollen wir eine gewünschte Frequenz angeben können. Nach dieser Frequenz soll dann der Delay zwischen dem Ein- und

Ausschalten der LED berechnet werden. Dieses Mal wollen wir die große High-Power LED nutzen.  
Legt dazu eine Variable `frequenz` sowie eine Variable `periode` an. Nutzt dann die Variable `periode` in den `delay()` Funktionen.  
Lasst dann die High-Power LED (Pin 4) flimmern.  
Möglichlicherweise braucht ihr für diese Aufgabe einen neuen Datentypen:

- `double`: Hält eine Komma-Zahl. Mathematische Kommas werden mit Punkten geschrieben! (Zum Beispiel 2.5)

## 10 Referenz

### 10.1 C-Syntax

- Variablen Deklaration

```
TYP name = Wert;
```

Hier wird eine Variable vom Typ TYP, mit Namen name und dem Wert Wert deklariert. Der Typ muss nur bei der ersten Zuweisung dabei stehen. Bei jeder weiteren Zuweisung muss der Typ weggelassen werden. Also zum Beispiel

```
= Wert2;
```

Weiteres Beispiel:

```
1 int counter = 0;  
2 ...  
3 counter = 3;
```

- IF-Bedingung

```
if( Bedingung ) { } else { }
```

Hier findet eine Verzweigung des sogenannten Kontrollflusses statt. Das heißt, dass nur einer der angegebenen Pfade ausgeführt wird. Wenn die Bedingung zutrifft, wird der Pfad im ersten Set gesweifter Klammern ausgeführt. Ist sie jedoch nicht erfüllt, so wird der Pfad im zweiten Set geschweifter Klammern nach dem else ausgeführt. Der zweite Pfad ist hierbei optional.

Beispiel:

```
1 boolean kontrolle = false;  
2 ...  
3 if(kontrolle == true) {  
4     digitalWrite(LED_BUILTIN, HIGH); //Dieser Pfad wird nicht ausgeführt, da kontrolle den Wert  
5         false hat, und somit nicht true ist.  
6 } else {  
7     digitalWrite(LED_BUILTIN, LOW); //Dieser Pfad ist der, der ausgeführt wird  
8 }  
9 ...  
10 boolean kontrolle2 = false;  
11 ...  
12 if(kontrolle2 == true) {  
13     digitalWrite(LED_BUILTIN, LOW); //Hier wird dieser Pfad nicht ausgeführt, da kontrolle2 den  
14         Wert false, also nicht den Wert true hat.  
15 } //Da allerdings keine "else"-Pfad existiert, wird er einfach übersprungen.
```

- While-Schleife

```
while(Bedingung){ }
```

While-Schleifen wiederholen den in ihnen enthaltenen Code solange die Bedingung erfüllt ist. Beispiel:

```
1 int counter = 3;  
2 ...  
3 while(counter > 0) { //Hier wird geprüft, ob die Variable counter größer als 0 ist.  
4     counter = counter - 1; //Dadurch wird von der Variable counter solange 1 abgezogen, bis sie  
5         kleiner/gleich 0 ist.  
6 }
```

- For-Schleife

```
for(Definition; Bedingung; Anweisung){ }
```

For-Schleifen definieren eine Variable, die sich bei jeder Wiederholung entsprechend der Anweisung solange die Bedingung erfüllt ist. Beispiel:

```
1 for(int counter = 0; counter < 10; counter++) { //Diese Schleife wird so oft ausgeführt bis  
2     counter größer/gleich 10 ist. Bei jeder Wiederholung counter um 1 erhöht durch "counter++".  
3     Serial.println("Hallo"); //In jeder Wiederholung wird dieser Code ausgeführt  
4 }
```

---

## 10.2 Datentypen

---

- **int**  
Integers (oder ints) bezeichnen ganze Zahlen.
- **double**  
Doubles halten Kommazahlen. Sie werden hier allerdings nicht benötigt werden.
- **long**  
Longs beschreiben sehr große Zahlen. Die Arduino Befehle `millis()` oder `micros()` geben beispielsweise einen Long zurück.
- **bool**  
Booleans stellen Wahrheitswerte dar und können entweder `true`, also wahr oder `false`, also falsch sein.
- **String**  
Strings repräsentieren Text. Dieser Text muss in Anführungszeichen stehen.
- **const**  
Consts bezeichnen Konstanten, die während des Programmverlaufs nicht geändert werden können. Sie benötigen noch einen weiteren Datentyp um zu spezifizieren, welche Daten wie speichern sollen.

---

## 10.3 Arduino Befehle

---

- **void setup()**  
Diese Funktion wird als Initialisierung einmal zu Beginn des Programms durchlaufen. Hier können die Ein- und Ausgänge des Controllers definiert werden
- **void loop()**  
Diese Funktion wird zur Laufzeit des Programmes ständig in einer Endlosschleife aufgerufen. Hier sollte die eigentliche Logik der Anwendung implementiert werden.
- **void pinMode(pin, mode)**  
Konfiguriert das Verhalten eines Pins, der Modus kann entweder INPUT oder OUTPUT sein.
- **void digitalWrite(pin, value)**  
Legt den Wert HIGH oder LOW auf einen Pin.
- **int digitalRead(pin)**  
Liest den Zustand eines Pins aus und gibt diesen zurück.
- **void analogWrite(pin, value)**  
Schreibt einen analogen Wert zwischen 0 und 255 als PWM-Signal auf einen Ausgang.
- **void delay(millis)**  
Wartet den übergebenen Wert in Millisekunden.
- **void delayMicroseconds(micros)**  
Wartet den übergebenen Wert in Mikrosekunden.

---

## 11 Kontakt

---

Jonas Kobbert: [kobbert@lichttechnik.tu-darmstadt.de](mailto:kobbert@lichttechnik.tu-darmstadt.de)

Sebastian Beck: [beck@lichttechnik.tu-darmstadt.de](mailto:beck@lichttechnik.tu-darmstadt.de)

Julian Euler: [julian.euler@stud.tu-darmstadt.de](mailto:julian.euler@stud.tu-darmstadt.de)

---

## 12 Links

---

VLC Bibliothek: <https://github.com/Lithimlin/VLC-Transceiver>

Adafruit\_GFX Bibliothek: <https://github.com/adafruit/Adafruit-GFX-Library>

Max72xxPanel Bibliothek: <https://github.com/Lithimlin/arduino-Max72xxPanel>

Timer Bibliothek: <https://github.com/JChristensen/Timer/archive/v2.1.zip>