# Arbeiten in der Arduino-Programmierumgebung

Deine ersten beiden Arduino-Programme im Rahmen dieses Arbeitspakets haben so ausgesehen:

```
void setup() {
  pinMode(5,OUTPUT);
  digitalWrite(5,LOW);
}

void loop() {
  digitalWrite(5,HIGH);
  delay(2000);
  digitalWrite(5,LOW);
  delay(5000);

}
```

```
int ledPin = 5;

void setup() {
  pinMode(ledPin,OUTPUT);
  digitalWrite(ledPin,LOW);
}

void loop() {
  digitalWrite(ledPin,HIGH);
  delay(2000);
  digitalWrite(ledPin,LOW);
  delay(5000);

}
```

Mit beiden Programmen hast du eine Schaltung „gesteuert“, bei der die Spannung über den **Steckkontakt** auf dem Arduino-Board mit der **Beschriftung „~5“** zur Verfügung gestellt wurde – dieser Pin wurde ja auch mit dem längeren Beinchen der Leuchtdiode verbunden, und dieses längere Beinchen muss ja stets mit dem positiven Pol der Spannungsquelle verbunden sein.

Damit dies funktioniert, muss festgelegt werden, dass dieser Steckkontakt als positiver Pol dienen kann – dies erfolgt über den Programmbefehl

```
pinMode(5, OUTPUT);
```

(beachte den **Strichpunkt**, der in dieser Programmierumgebung am Ende eines Programmbefehls vorhanden sein muss, und beachte auch die **Groß- und Kleinschreibung**, die strikt eingehalten werden muss!).

Ebenso könnte ein anderer Steckkontakt in dieser Reihe als positiver Pol verwendet werden, z.B. der Steckkontakt mit der Beschriftung „~3“. Natürlich müsste dann der Programmbefehl entsprechend `pinMode(3, OUTPUT);` lauten. Die Zahl in dieser Beschriftung bezeichnen wir als die **Portnummer**.

Der Vorteil bei der Verwendung dieser Steckkontakte als positiver Spannungspol im Unterschied zu dem mit **5V** beschrifteten Steckkontakt ist, dass bei diesen die Spannung über Programmbefehle „ein- und ausgeschaltet“ werden kann.

Der Befehl

```
digitalWrite (5, HIGH);
```

bewirkt, dass am Steckkontakt mit der Beschriftung „~5“ positive Spannung zur Verfügung steht, der Befehl

```
digitalWrite(5, LOW);
```

hingegen, dass an diesem Steckkontakt keine Spannung (entsprechend 0 V) zur Verfügung steht.

Umgangssprachlich würde man sagen, dass mit diesen beiden Befehlen „die Elektrizität ein- bzw. ausgeschaltet wird“.

CC BY-NC-SA 4.0 Informatik-Werkstatt AAU 2021

Die beiden Programme haben noch etwas **gemeinsam**:
Beide Programme bestehen im Wesentlichen aus jeweils **zwei Programmteilen**, einem, der mit `setup`() überschrieben ist, und einem, der mit `loop`() überschrieben ist.

Im `setup`-Programmteil wird festgelegt, welche Einstellungen bei Programmstart gelten sollen. Dieser Programmteil wird nur ein einziges Mal, eben bei Programmstart nachdem das Programm auf das Arduino-Board übertragen wurde, ausgeführt.

Der `loop`–Programmteil hingegen wird immer wieder ausgeführt, solange das Arduino-Board z.B. über ein USB-Kabel vom Computer mit Spannung versorgt wird. Der Befehl (z.B.)

```
delay(4500);
```

sorgt dabei dafür, dass zwischendurch eine Pause (hier von 4500 Millisekunden, also 4,5 Sekunden) gemacht wird, bis der jeweils nächste Befehl ausgeführt wird.

Und was auch wichtig ist:
Jeder der beiden Programmteile wird durch **ein Paar geschwungener Klammern { }** begrenzt.

Es gibt aber auch einen wesentlichen **Unterschied** zwischen den beiden Programmen. Während im ersten Programm der Steckkontakt direkt über seine Portnummer angesprochen wird, also z.B.

```
pinMode(5, OUTPUT);,
digitalWrite(5, HIGH);,
```

wird im zweiten Programm eine sogenannte **Variable** namens `ledPin` verwendet, in der die Portnummer „gemerkt“ wurde. In diesem Fall lauten die entsprechenden Befehlszeilen:

```
pinMode(ledPin, OUTPUT);,
digitalWrite(ledPin, HIGH);.
```

Damit eine **Variable** verwendet werden kann, muss sie aber (am besten gleich zu Beginn) **vereinbart** (man sagt auch: **deklariert**) werden. Dies erfolgt im zweiten Programm mit der Befehlszeile

```
int ledPin = 5;
```

Eine **Variablendeklaration** besteht aus zumindest drei Teilen,

- einem **Schlüsselwort** zu Beginn, das angibt, welche Werte in der Variablen gemerkt werden können (im Beispielprogramm zeigt `int` an, dass in der Variablen `ledPin` Werte vom **Datentyp Integer** (das sind **ganze Zahl**en) gemerkt werden können.
- dem **Namen** der Variablen, der immer dann in Programmbefehlen verwendet werden kann, wenn der in der Variablen gespeicherte Wert benötigt wird (im Beispielprogramm ist dieser Name `ledPin`),
- einem **Strichpunkt am Ende** der Variablendeklaration.
- Zusätzlich kann – wie im Beispielprogramm – dieser Variablen auch gleich ein Wert zugewiesen werden. Dies erfolgt durch `ledPin = 5;` (lies: „die Variable namens `ledPin` kriegt den Wert `5`“).

Beachte, dass diese drei (vier) Teile einer **Variablendeklaration** genau in dieser Reihenfolge geschrieben werden müssen!

CC BY-NC-SA 4.0 Informatik-Werkstatt AAU 2021

Wir verwenden für **Variable** folgende Modellvorstellung:

Eine **Variable verweist auf einen Bereich im Speicher** des Computers, in dem ein bestimmtes Datum („ein Wert“) gespeichert wird. Anders gesagt: Eine **Variable** gibt einem **Bereich im Speicher** des Computers einen **Namen**.

Diese Bereiche des Computerspeichers können wir uns zum Beispiel so vorstellen:

| | | | | | |
|---|---|---|---|---|---|
| 10010011 | 11100101 | 11110100 | 00010111 | 10100110 | 00110010 |
| 10110010 | 10000111 | 10010110 | 11010101 | 10101111 | 10110101 |
| 11111111 | 00000000 | 11000100 | 11010101 | 11100110 | 00111110 |
| 11111111 | 00000000 | 11000100 | 11010101 | 11100110 | 00111110 |

Die in den Bereichen gespeicherten Daten sind als **Bitmuster**, also durch eine **Folge der Ziffern Null und/ oder Eins** dargestellt – das hat damit zu tun, dass der Computer nur zwei Zustände unterscheiden kann: „Spannung liegt an“ (codiert durch „1“) oder „keine Spannung liegt an“ (codiert durch „0“). Du darfst dir einen solchen Speicherbereich als eine Folge von – in obiger Abbildung acht – Schaltern vorstellen, die entweder „ein“ (dargestellt durch „1“)oder „aus“ (dargestellt durch „0“) sind.

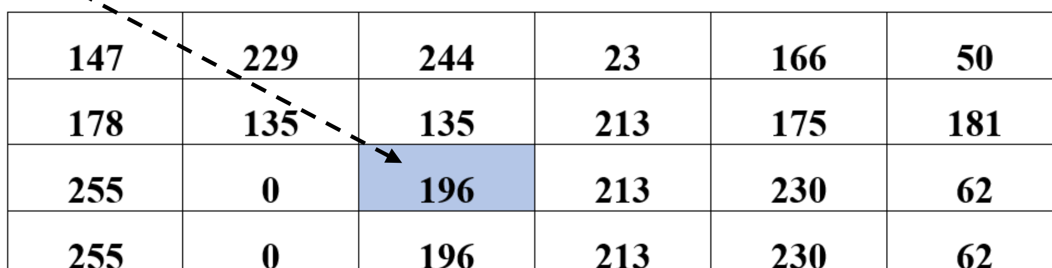
Zur Vereinfachung nehmen wir zunächst an, dass die dargestellten Daten allesamt ganze Zahlen, also vom **Datentyp Integer** (`int`) sind. Dann sieht die Abbildung von den Bereichen des Speichers so aus:

| | | | | | |
|---|---|---|---|---|---|
| 147 | 229 | 244 | 23 | 166 | 50 |
| 178 | 135 | 135 | 213 | 175 | 181 |
| 255 | 0 | 196 | 213 | 230 | 62 |
| 255 | 0 | 196 | 213 | 230 | 62 |

(wie du aus einem Bitmuster die entsprechende ganze Zahl berechnen kannst, kannst du im Anhang dieser Informationsdatei erfahren).

Bei der **Variablendeklaration** wird ein **passender Speicherbereich** für die betreffende Variable **reserviert**, z.B.:

```
int ledPin;
```

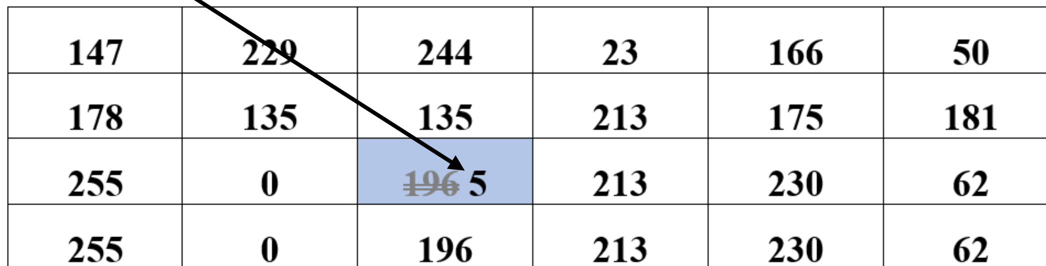


| | | | | | |
|---|---|---|---|---|---|
| 147 | 229 | 244 | 23 | 166 | 50 |
| 178 | 135 | 135 | 213 | 175 | 181 |
| 255 | 0 | 196 | 213 | 230 | 62 |
| 255 | 0 | 196 | 213 | 230 | 62 |

Nach dieser **Variablendeklaration** kann daher über die **Variable** namens `ledPin` auf den **Inhalt des** blau markierten **reservierten Speicherbereichs** zugegriffen werden.

Da zunächst im betreffenden Speicherbereich aber irgendein Bitmuster gespeichert wird, muss der im Programm benötigte **Wert über** eine Wertzuweisung an **die Variable in diesen Speicherbereich geschrieben** werden. Dies erfolgt in unserem Beispiel einfach durch

```
ledPin = 5;
```

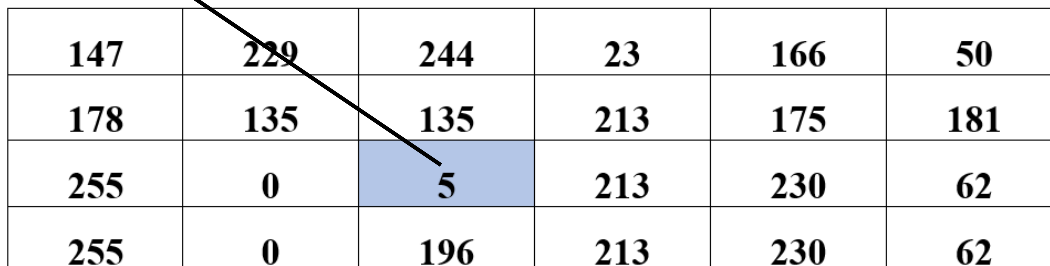


| 147 | 229 | 244 | 23 | 166 | 50 |
|---|---|---|---|---|---|
| 178 | 135 | 135 | 213 | 175 | 181 |
| 255 | 0 | ~~196~~ 5 | 213 | 230 | 62 |
| 255 | 0 | 196 | 213 | 230 | 62 |

Der vorher in diesem Speicherbereich gespeicherte Wert wird dabei überschrieben und geht daher „verloren“.

Wenn dieser gespeicherte Wert im Programm wieder benötigt wird, kann er im reservierten Speicherbereich über die Variable auch wieder gelesen werden. Dies erfolgt z.B. beim Befehl

```
pinMode(ledPin, OUTPUT);
```



| 147 | 229 | 244 | 23 | 166 | 50 |
|---|---|---|---|---|---|
| 178 | 135 | 135 | 213 | 175 | 181 |
| 255 | 0 | 5 | 213 | 230 | 62 |
| 255 | 0 | 196 | 213 | 230 | 62 |

In unserem Beispiel wird dabei also der Wert „5“ im Speicher gelesen und für die Variable `ledPin` eingesetzt – das Programm „versteht“ daher den Befehl als `pinMode(5, OUTPUT);`

Beachte aber, dass der Wert beim Lesen im Speicherbereich verbleibt – auch **nach dem Lesen im Speicherbereich bleibt dort** also **der Wert** 5 **gespeichert**!

CC BY-NC-SA 4.0 Informatik-Werkstatt AAU 2021

## Anhang – Umwandlung einer Bitfolge in die entsprechende ganze Zahl

Wie findet man heraus, dass z.B. die Bitfolge 11110100 der (dezimalen) Zahl 244 entspricht?

Beginnen wir unsere Überlegungen mit der Zahl 244!

Du weißt sicherlich, dass „244“ eine abkürzende Schreibweise für $2 \cdot \mathbf{100} + 4 \cdot \mathbf{10} + 4 \cdot \mathbf{1}$ ist:

Die Ziffer ganz rechts gibt an, wie oft der Wert **1** zur Zahl beiträgt,

die nächste Ziffer links gibt an, wie oft der Wert **10** zur Zahl beiträgt,

die nächste Ziffer links gibt an, wie oft der Wert **100** zur Zahl beiträgt usw.

Bei dieser Zahlendarstellung mit den **zehn Ziffern** 0, 1, 2, 3, 4, 5, 6, 7, 8 und 9 ver**zehn**facht sich also der Wert, den die Ziffer darstellt, wenn wir um eine Stelle „nach links“ weitergehen, man spricht vom **Stellenwertsystem zur Basis 10** bzw. von **dezimalen** Zahlen (decem, lat.: zehn).

Wenn wir nun, wie bei einer **Bitfolge**, nur die **zwei Ziffern** 0 und 1 zur Verfügung haben, können wir analog vorgehen:

Die Ziffer ganz rechts gibt an, wie oft der Wert **1** zur Zahl beiträgt,

die nächste Ziffer links gibt an, wie oft der Wert **2** zur Zahl beiträgt,

die nächste Ziffer links gibt an, wie oft der Wert **4** zur Zahl beiträgt usw.,

d.h. hier verdoppelt („verzweifacht“) sich der Wert, den die Ziffer darstellt, wenn wir um eine Stelle „nach links“ weitergehen“ – man spricht vom **Stellenwertsystem zur Basis 2** bzw. von **dual**en Zahlen.

Die Bitfolge 11110100 lässt sich dann also so interpretieren:

$11110100 = 1 \cdot \mathbf{128} + 1 \cdot \mathbf{64} + 1 \cdot \mathbf{32} + 1 \cdot \mathbf{16} + 0 \cdot \mathbf{8} + 1 \cdot \mathbf{4} + 0 \cdot \mathbf{2} + 0 \cdot \mathbf{1}$, entsprechend der dezimalen Zahl 244.

…und noch ein Tipp: Damit es bei Verwendung von Stellenwertsystemen mit unterschiedlicher Basis nicht zu Irrtümern kommt, schreibt man die Zahl üblicherweise mit der Basis des Stellenwertsystems an. In unserem Beispiel sollte man also das Ergebnis der Umrechnung so anschreiben:

$$11110100_{\mathbf{2}} = 244_{\mathbf{10}}.$$