Arduino Einsteigerkurs – Teil 2

Arduino IDE & Einführung in die Programmierung

In diesem Kursteil lernst du die Arduino IDE kennen und schreibst deine ersten eigenen Programme.

Um einen Arduino zu programmieren, benötigt man ein spezielles Programm auf dem Computer.
Dieses Programm heißt **Arduino IDE**.

Die Arduino IDE ist die **Arbeitsumgebung**, in der:

- Programme geschrieben werden
- Fehler überprüft werden
- Programme auf den Arduino übertragen werden

Ohne die Arduino IDE kann der Arduino **nicht ohne sehr tiefes Wissen und Spezialhardware programmiert** werden.

---

1 Was ist die Arduino IDE?

Die Arduino IDE ist ein Programm, das auf dem Computer installiert wird
(zum Beispiel unter Windows, macOS oder Linux).

In der Arduino IDE:

- schreibst du den **Programmcode**
- prüfst du, ob der Code **fehlerfrei** ist
- lädst du das Programm auf den Arduino hoch

🧠 **Merksatz:**

Die Arduino IDE ist die Verbindung zwischen Computer und Arduino.

---

2 Der Quelltext (Sketch)

Das Programm, das du in der Arduino IDE schreibst, nennt man **Sketch**.

Ein Sketch ist:

- der **Quelltext** deines Programms
- eine Abfolge von Anweisungen
- die Beschreibung dessen, was der Arduino tun soll

Beispiel:

- LED einschalten
- warten
- LED ausschalten

Der Sketch wird in der Programmiersprache **C++** geschrieben
(in einer für Anfänger vereinfachten Form).

📌 Wichtig:

Der Arduino versteht **keinen Text**, sondern nur **Maschinencode**.
Die Arduino IDE übersetzt den Sketch für den Arduino.

---

3 Überprüfen des Programms

Bevor ein Programm auf den Arduino geladen wird, sollte es **überprüft** werden.

In der Arduino IDE gibt es dafür den Button **„Überprüfen“** (Häkchen).

Beim Überprüfen:

- wird der Sketch **übersetzt**
- werden **Syntaxfehler** gesucht (z. B. fehlendes Semikolon)
- wird geprüft, ob das Programm grundsätzlich lauffähig ist

🧠 **Merksatz:**

Überprüfen heißt: „Ist mein Programm korrekt geschrieben?“

---

4 Hochladen des Programms

Nach dem Überprüfen kann der Sketch auf den Arduino **hochgeladen** werden.

Dazu dient der Button **„Hochladen“** (Pfeil nach rechts).

Beim Hochladen:

- wird das Programm über das USB-Kabel übertragen
- wird der alte Programmcode auf dem Arduino ersetzt
- startet das neue Programm automatisch

📌 Wichtig:

Nach dem Hochladen läuft das Programm **direkt auf dem Arduino** –
auch wenn der Computer später getrennt wird.

---

5 Arduino führt das Programm aus

Sobald das Programm erfolgreich hochgeladen wurde:

- startet der Arduino neu
- führt zuerst `setup()` aus
- danach läuft `loop()` endlos

Der Arduino arbeitet nun **selbstständig**.

🧠 **Merksatz:**

Der Computer programmiert –
der Arduino führt aus.

🧠 Zusammenfassung

- Die Arduino IDE ist das **Werkzeug zum Programmieren**
- Ein Arduino-Programm heißt **Sketch**
- „Überprüfen“ sucht Fehler im Code
- „Hochladen“ überträgt das Programm auf den Arduino
- Der Arduino führt das Programm **eigenständig** aus

👉 **Die Arduino IDE ist der erste und wichtigste Schritt auf dem Weg zum eigenen Arduino-Projekt.**

1. Aufbau eines Arduino-Programms

Grundstruktur eines Arduino-Programms: `setup()` und `loop()`

Jedes Arduino-Programm folgt **immer der gleichen Grundstruktur**.
Egal, ob es sich um ein kleines Blink-Programm oder ein größeres Projekt handelt – **diese zwei Bestandteile gibt es immer**.

---

1 Die Funktion `setup()`

```
void setup() {
  // Code hier
}
```

Was ist `setup()`?

`setup()` ist ein Programmteil (eine sogenannte **Funktion**), der:

- **genau einmal** ausgeführt wird
- **beim Start des Arduino** läuft
- direkt nach dem Einschalten oder Hochladen des Programms

Man kann sich `setup()` vorstellen wie:

das **Vorbereiten** des Arduino, bevor die eigentliche Arbeit beginnt.

Was passiert typischerweise in `setup()`?

In `setup()` werden Dinge eingerichtet, zum Beispiel:

- Pins als **Ein- oder Ausgang** festlegen (`pinMode`)
- serielle Kommunikation starten (`Serial.begin`)
- Startwerte für Variablen setzen
- Displays, Sensoren oder Module initialisieren

Merksatz:

`setup()` bereitet alles vor, was das Programm später braucht.

---

2 Die Funktion `loop()`

```
void loop() {
  // Code hier
}
```

Was ist `loop()`?

`loop()` ist der Teil des Programms, der:

- **endlos wiederholt** wird
- immer wieder **von vorne beginnt**
- solange läuft, wie der Arduino eingeschaltet ist

Man kann sich `loop()` vorstellen wie:

eine **Dauerschleife**, die niemals endet.

Was passiert typischerweise in `loop()`?

In `loop()` stehen alle Anweisungen, die:

- ständig überprüft werden sollen
- immer wieder ausgeführt werden müssen

Zum Beispiel:

- Taster abfragen
- Sensorwerte lesen
- LEDs schalten
- Zeit prüfen (`millis()`)

🧠 **Merksatz:**

Alles, was der Arduino **immer wieder tun soll**, gehört in `loop()`.

---

3 Zusammenspiel von `setup()` und `loop()`

Der Ablauf eines Arduino-Programms ist immer gleich:

1. Arduino startet
2. `setup()` wird **einmal** ausgeführt
3. `loop()` startet
4. `loop()` läuft **immer wieder von vorne**
5. Das Programm endet **nie von selbst**

📌 Wichtig:

Es gibt **kein Ende** eines Arduino-Programms –
es läuft, solange der Arduino Strom hat.

---

4 Bildliche Vorstellung

`setup()` ist wie:

morgens alles vorbereiten (Licht an, Kaffee kochen)

- `loop()` ist wie:

  den ganzen Tag immer wieder die gleiche Aufgabe erledigen

🧠 Zusammenfassung

- Jedes Arduino-Programm hat **immer** `setup()` und `loop()`
- `setup()` läuft **einmal beim Start**
- `loop()` läuft **endlos**
- Diese Struktur ist die Grundlage für **alle Arduino-Projekte**

👉 **Wer `setup()` und `loop()` verstanden hat, hat das Herz eines Arduino-Programms verstanden.**

Das erste Programm: LED blinken

Das klassische erste Programm lässt eine LED blinken.

```
pinMode(Pin, OUTPUT);  Pin vorbereiten
digitalWrite(Pin, HIGH); LED an
delay(1000); warten
digitalWrite(Pin, LOW)M LED aus
```

Blink-LED am Arduino – zwei Wege, ein Ziel

Zu Beginn arbeiten wir mit dem klassischen **Blink-Programm**.
Dabei wird die interne LED des Arduino Uno (Pin 13) im Sekundentakt ein- und ausgeschaltet. Dieses Beispiel ist bewusst sehr einfach gehalten und eignet sich hervorragend, um erste Grundkonzepte kennenzulernen.

In der ersten Version verwenden wir die Funktion `delay()`:

Der Arduino schaltet die LED ein, wartet eine Sekunde, schaltet sie aus und wartet erneut eine Sekunde.
Das Programm ist leicht zu lesen und sofort verständlich – **aber es hat einen wichtigen Nachteil**.

Während `delay()` läuft, **steht der komplette Arduino still**.
Er kann in dieser Zeit nichts anderes tun:

- keine Taster abfragen
- keine Sensoren lesen
- keine seriellen Daten verarbeiten

Der Arduino „schläft“ während der Wartezeit.
Für erste Experimente ist das in Ordnung, für echte Projekte aber problematisch.

Man kann sich `delay()` vorstellen wie:

„Ich mache jetzt eine Pause und schließe die Augen.“

Warum das in der Praxis ein Problem ist

Sobald ein Programm mehr als **eine Aufgabe gleichzeitig** erledigen soll, kommt `delay()` an seine Grenzen.
Wenn zum Beispiel:

- eine LED blinken soll
- **und** gleichzeitig ein Taster reagieren soll

dann kann der Arduino den Tastendruck während der Wartezeit **verpassen**.
Das Programm reagiert träge oder gar nicht.

Deshalb lernen wir eine zweite, bessere Lösung kennen.

Die Blink-LED mit `millis()`

In der zweiten Version wird die LED ebenfalls im Sekundentakt geschaltet – **aber ohne `delay()`.**

Statt zu warten, fragt der Arduino mit `millis()` ständig die aktuelle Zeit ab.
`millis()` liefert die Anzahl der Millisekunden seit dem Start des Arduino.

Das Programm merkt sich:

- **wann** die LED zuletzt umgeschaltet wurde
- **wie viel Zeit** vergehen soll

In jeder Runde der `loop()`-Funktion wird geprüft:

Ist seit dem letzten Umschalten genug Zeit vergangen?

Nur dann wird die LED umgeschaltet.
Ansonsten läuft das Programm einfach weiter.

Der entscheidende Unterschied:
Der Arduino **wartet nicht** – er **arbeitet durchgehend weiter**.

Man kann sich `millis()` vorstellen wie:

„Ich schaue ständig auf die Uhr, während ich weiterarbeite."

Direkter Vergleich

Blink mit `delay()`	Blink mit `millis()`
sehr einfach	etwas komplexer
gut für Einsteiger	ideal für echte Projekte
Arduino wartet	Arduino arbeitet weiter
blockiert alles	nichts blockiert
schlecht für mehrere Aufgaben	perfekt für Multitasking

Wichtiger Merksatz für den Kurs

`delay()` ist gut zum Lernen –
aber schlecht für echte Programme.

Oder noch einfacher:

Ein Arduino sollte nicht warten,
er sollte denken.

Fazit

Beide Programme lassen die LED blinken –
aber nur die `millis()`-Version ist **zukunftssicher**.

Wer `millis()` verstanden hat, kann:

- mehrere LEDs unabhängig steuern
- Taster zuverlässig abfragen
- Sensoren auslesen
- serielle Kommunikation nutzen

Damit ist `millis()` ein ganz wichtiger Schritt
vom **Spielbeispiel** hin zu **echten Arduino-Projekten**.

Variablen & Logik

In einem Arduino-Programm (und generell in der Programmierung) müssen wir uns **Werte merken können**.
Dafür gibt es **Variablen**.

◈ Was ist eine Variable?

Eine Variable ist ein **Speicherplatz im Arduino**, der:

- einen **Namen** hat
- einen **Wert** enthält
- und dessen Wert sich **während des Programms ändern kann**

Man kann sich eine Variable vorstellen wie:

eine **beschriftete Kiste**, in der ein Wert liegt.

◈ Eine Variable anlegen

```
int zaehler = 0;
```

Das bedeutet:

- `int` → Der Datentyp
  - steht für **ganze Zahlen** (z. B. 0, 1, 2, 10, -5)
- `zaehler` → Der Name der Variable
  - frei wählbar, sollte **verständlich** sein
- `= 0` → Der Startwert
  - der Zähler beginnt bei **0**
- `;` → Jedes Arduino-Kommando endet mit einem Semikolon

🧠 **Merksatz:**

Eine Variable braucht immer **Typ, Namen und Startwert**.

◈ Warum heißt die Variable „Zähler“?

Der Name `zaehler` deutet an, **wofür** die Variable benutzt wird:
Sie soll **mitzählen**, also ihren Wert verändern.

Zum Beispiel:

- wie oft eine Taste gedrückt wurde
- wie oft eine LED geblinkt hat
- wie viele Sekunden vergangen sind

◈ Den Wert einer Variable ändern

```
zaehler = zaehler + 1;
```

Das sieht auf den ersten Blick vielleicht verwirrend aus, bedeutet aber:

„Nimm den aktuellen Wert von `zaehler`
und erhöhe ihn um 1.“

Beispiel:

- vorher: `zaehler = 0`
- danach: `zaehler = 1`
- beim nächsten Mal: `zaehler = 2`

Der Zähler **zählt hoch**.

🧠 **Merksatz:**

Eine Variable kann ihren eigenen alten Wert verwenden,
um einen neuen Wert zu berechnen.

◈ Warum ist das wichtig?

Mit Variablen kann ein Programm:

- sich Dinge **merken**
- auf **Ereignisse reagieren**
- **Entscheidungen treffen**

Ohne Variablen:

- würde ein Programm immer gleich ablaufen
- gäbe es kein Zählen, keine Zeitmessung, keine Zustände

◈ Verbindung zur Logik im Programm

Variablen werden oft zusammen mit **Logik** verwendet, zum Beispiel:

```
if (zaehler > 10) {
  // etwas passiert
}
```

Das bedeutet:

„Wenn der Zähler größer als 10 ist, dann tue etwas.“

So kann der Arduino **Entscheidungen treffen**.

🧠 Zusammenfassung

- Variablen speichern **veränderliche Werte**
- Sie bestehen aus **Datentyp, Name und Wert**
- Der Wert einer Variable kann im Programm geändert werden
- Variablen sind die Grundlage für:
    - Zähler
    - Zeitmessungen
    - Zustände
    - Entscheidungen

👉 Mit Variablen und einfacher Logik kannst du bereits viele Arduino-Programme verstehen, verändern und erweitern.

Bei Fragen einfach den Kursleiter fragen 😊