Table of contents

Table of contents

1. Preface and didactic considerations

Getting started is always difficult - but not with this guide for Arduino. It is intended to serve as a foundation for learning the Arduino platform and provide beginners with a didactically sound, simple, interesting, and closely guided introduction to Arduino topics. Competence acquisition through this booklet covers various areas of microcontroller programming so broadly that the reader will be enabled to independently delve into advanced topics. This includes learning additional programming possibilities or using additional modules.

This work was developed over a period of almost ten years in parallel with technical training for teachers and classroom use with students of various age groups. Over the years, it has been evaluated, updated, and expanded multiple times. The result is a workbook that can be used in conjunction with teaching or courses.

Use in the classroom

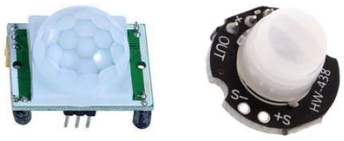
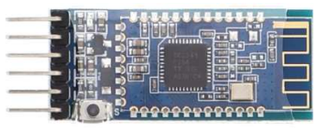
The **didactic concept** of this guide is based on the learners **working** largely **independently** with the instructions and corresponding electronic components. The course instructor provides the necessary materials for each instruction and then only offers **guidance** to learners, for example, to support individual participants or small groups. Especially in the initial phase, learners may struggle to identify errors in electronics, programming syntax, or program logic on their own. However, learners will quickly develop a sense of **problem-solving** in these areas. It is important in this regard to read the **theoretical introduction** before the practical exercises to avoid failure due to lack of basic knowledge during later practical tasks.

The **open working method** achieves optimal and **effective learning time** since **everyone can work and learn at their own pace**. Furthermore, with the completion of each instruction, a project-like group dynamic quickly develops, as each individual works independently but all pursue the same goal. Mutual explanation and assistance support the **learning process** of all participants and **promote teamwork**.
In many work phases, there are **opportunities for differentiation** by encouraging students to expand upon the examples provided or **creatively** incorporate their own ideas into solving the task.

1.1 Preparation and procurement of materials

The following materials are required for the exercises in this booklet:

01. Microcontroller (UNO MEGA /NANO) 	02. Breadboard 	03. Jumper Wire Male / Male 
04. Jumper Wire Female / Female 	05. Jumper Wire Female / Male 	06. Servo motor MG90S or SG90 
07. 20 LEDS each (green, red, yellow, blue, white) 	08. RGB-LED 	09. Infrared transmitter and receiver 
10. Four-digit 7-segment display 	11. Relay module 	12. Infrared remote control 
13. Stepper motor with driver board 	14. Droplet sensor 	15. Moisture sensor 
16. RFID-KIT 	17. 20 Resistors each (100,200,300,1K,10K Ohm) 	18. Ultrasonic sensor 

19. Bluetooth module HC-05 	20. Tilt Sensor 	21. Motion detector HC-SR501 or SR602 
22. Temperature sensor TMP36 	23. Photoresisto 	24. Rotary potentiometer 
25. Button (big) 	26. Button (small) 	27. Piezospeaker 
28. I²C LCD 	29. Diode 	30. Speaker 
31. Traffic light module 	32. WS2812 LED module with 8 LEDs 	33. Bluetoothmodule (alternative to HC-05) 
34. NTC-temperature sensor 	35. 5V Fan or DC electric motor 	36. TIP120 transistor 
37. Light barrier 	38. MQ135 Gas sensor 	

1.2 What is Arduino?

Arduino is an open-source electronics prototyping platform for flexible, easy-to-use hardware and software in the field of microcontroller programming. It is suitable for realizing exciting and spectacular projects in a short amount of time. Many of these projects can be found under the term "Arduino" on platforms like YouTube. It is primarily used by artists, designers, tinkerers, and hobbyists to bring creative ideas to life. However, Arduino development environments are also increasingly being used in schools, colleges, and universities to provide learners with a creative, exciting, and above all, easy entry point into the field of microcontroller programming. Topics such as "automation technology," "robotics," etc., can also be explored using the Arduino development environment.

1.3 Development - Arduino, Funduino, Genuino

The story of Arduino began in 2005 when the two "tinkerers" Massimo Banzi and David Cuartielles developed their first microcontroller board, and programmer David Mellis created the corresponding syntax, which is based on the programming languages C++, C, and Assembler. The project is licensed under Creative Commons, making Arduino largely an open-source platform, which significantly facilitated its dissemination and development.

Over the years, the founders of the Arduino platform (Arduino LLC) and the producers of the official Arduino boards (Arduino S.r.l) became embroiled in a dispute as both groups claimed the brand name "Arduino" for themselves. At that time, it could not be definitively determined which party was the rightful owner of the trademark, leading to a legal dispute with far-reaching consequences. The online platform split into several internet presences (www.arduino.org, www.arduino.cc), with both parties distributing their own microcontrollers, both under the same brand name "Arduino." In 2015, founding member M. Banzi introduced the brand "Genuino." Genuino was henceforth the designation for microcontroller boards intended to be sold outside the United States. It was not until the World Maker Faire in 2016 that Arduino LLC and Arduino S.r.l announced the merger of the warring parties under a newly formed Arduino Holding. Over the years, due to uncertainty about the name rights, many new brands and names of other manufacturers of Arduino-compatible microcontroller boards emerged. Based on the open-source foundation, while the boards are technically identical in most cases, they are not allowed to be labeled with the name "Arduino." These boards are then referred to as "Arduino clones" or "Arduino-compatible." For example, the German company Funduino GmbH emerged during this legally problematic period. The company developed extensive teaching materials and learning kits for the educational sector as early as its inception in 2010. As it was not possible to establish constructive cooperation with Arduino, the company has since been producing its own Arduino-compatible microcontroller boards.

Today, the term Arduino represents the definition of flexible, easy-to-use hardware and software in the field of microcontroller programming. Arduino is ideally suited for realizing spectacular projects. Due to its low acquisition costs, practical development environment, resulting didactic value, and nearly endless combination possibilities of countless sensors and actuators, the field of microcontroller programming is increasingly making its way into the global education sector.

2. Hardware and Software

The term Arduino is commonly used interchangeably to refer to both the different Arduino boards (hardware) as well as the programming environment (software).

2.1 Hardware

In addition to the microcontroller, sensors, and actuators, the basis for quick and flexible experimental setups requires jumper wires in conjunction with a breadboard. This saves time-consuming soldering work. Furthermore, LEDs are very suitable for verifying the signal output of the board.

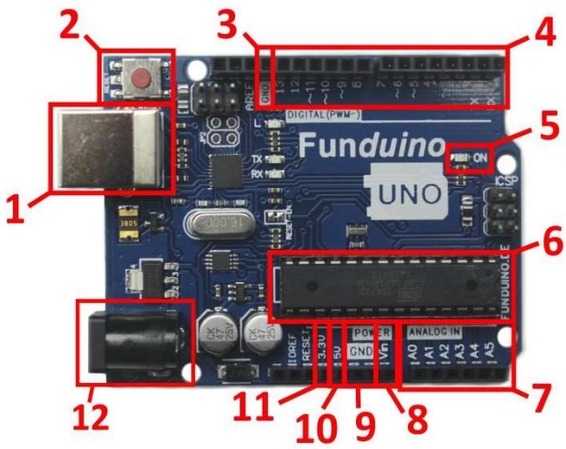
2.1.1 The microcontroller

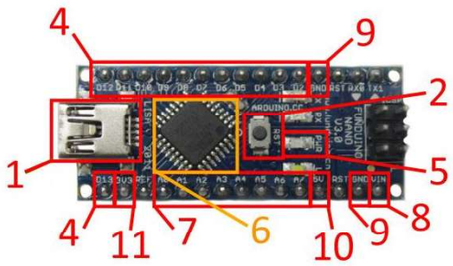
The "Arduino" is a type of microcontroller board (hereinafter referred to as "board"). It is a printed circuit board (PCB) with a variety of electronic components surrounding the actual microcontroller chip. Along the edge of the board, there are many slots (called pins) where various modules such as sensors and actuators can be connected. These include switches, LEDs, ultrasonic sensors, temperature sensors, rotary knobs, displays, motors, servos, etc.

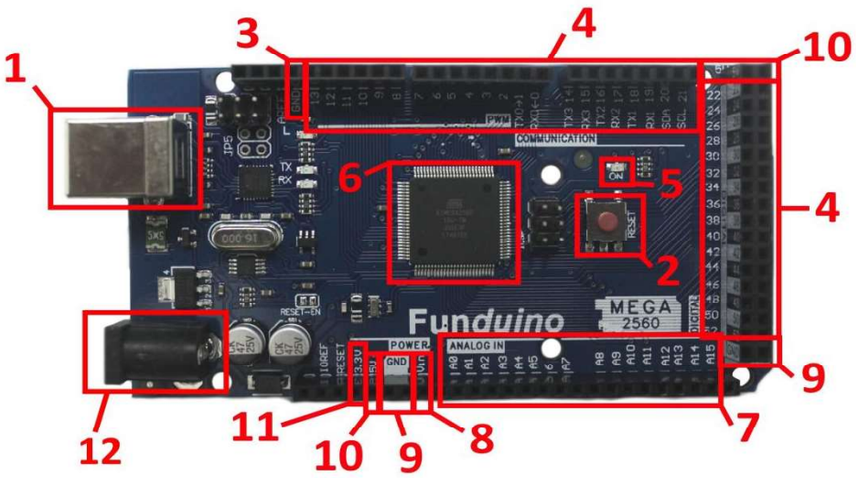
There are many different versions of microcontroller boards that can be used with the Arduino software. These include both many different large and small boards with the official "Arduino" designation as well as a variety of often cheaper Arduino-compatible boards. The boards differ only in small details, such as the number of digital pins or the amount of memory.

This guide was created using an Arduino-compatible UNO board from the Funduino brand. However, any Arduino-compatible controller can be used with this guide.

The following three boards are the most well-known microcontroller boards that can be used with Arduino software: UNO, NANO, and MEGA.





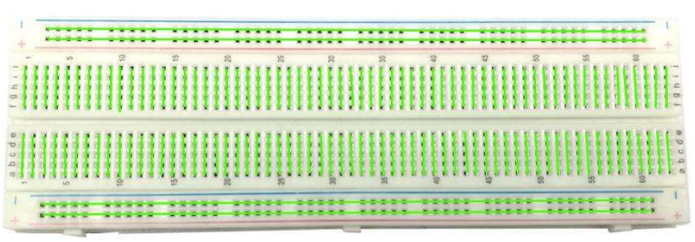


Legend:

1. USB connection	5. Power-ON indicator light	9. GND (*Ground or „-“*)
2. Reset button	6. Microcontroller	10. 5V output from the voltage regulator
3. GND (*Ground* or *„-“*)	7. Analog inputs	11. 3,3V output from the voltage regulator
4. Digital inputs and outputs	8. External power supply per pin *(Vin)*	12. External power supply (*7-12V*)

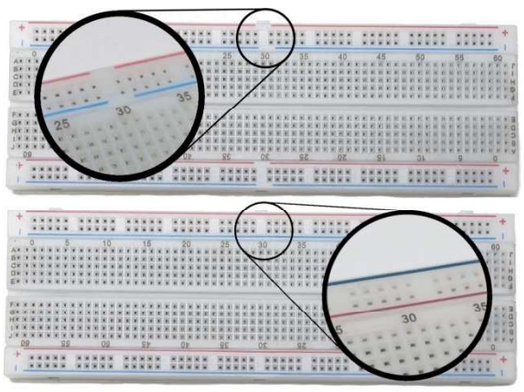
2.1.2 Accessories - The Breadboard

A breadboard, also known as a "prototyping board," is a useful tool for building circuits without the need for soldering. In a breadboard, multiple contacts are connected



to each other. Therefore, many connections can be made at these points without needing to be soldered or screwed together.

There are many different versions of breadboards available, ranging from large to small, colored to transparent. The wiring inside the breadboard can also vary. Generally, the contacts are connected as shown in the image. The outer lines are continuously connected to each other, and the small lines in the inner area are each connected to five slots vertically. The small inner segments are suitable for detailed constructions, while the outer lines are typically used for distributing power.



There are also breadboards where the outer contacts are subdivided in the middle or even multiple times. Therefore, before starting work, it is advisable to check how the available breadboard is connected. Often, the outer contacts are marked with red and blue lines, indicating whether the contacts are connected in a continuous line or not.

2.1.3 Accessories - LED

With LEDs, you can quickly test the results of a project. Therefore, they are useful for almost all Arduino projects.

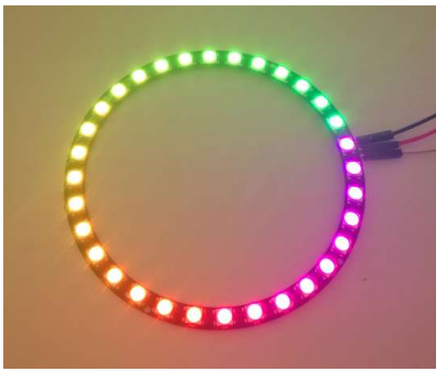
The most important information about light-emitting diodes (LEDs):

- The current can only flow through the LED in one direction. Therefore, it must be connected correctly. An LED has a longer and a shorter contact. The longer contact is "+" and the shorter one is "-".
- An LED is designed for a specific current. If this current is lower, the LED will glow less brightly or not at all. If the current is exceeded, the LED will quickly burn out and become hot at the contacts (Caution, risk of burns!). Excessive current may occur if the maximum voltage for the LED is exceeded. For example, if you connect an LED directly to the 5V output, it will be immediately damaged. Therefore, when using LEDs with Arduino boards, always use a current-limiting resistor.
- Typical voltage values for LED colors: Blue: 3.1V, White: 3.3V, Green: 3.7V, Yellow: 2.2V, Red: 2.1V. The exact specified voltage can be found in the datasheet for each LED.

- Unbinding recommendation for resistors when using the following LED colors at the 5V pins of the microcontroller:

LED-color:	white	red	yellow	green	blue	infrared-LED
Resistor:	100 Ohm	200 Ohm	200 Ohm	100 Ohm	100 Ohm	100 Ohm

Smart LEDs - NeoPixel



In addition to standard LEDs, there is also the "luxury version" in the form of WS2811, WS2812, and WS2812B LEDs. These LEDs are often referred to as NeoPixels. They are colored (RGB) LEDs with an integrated chip that controls the RGB LEDs. The advantage is that the LED is controlled with only three instead of four contacts, and many LEDs can be daisy-chained together without the need for additional separate cables. Pre-made WS2812 LED combinations are often found in the form of LED rings or colored LED light strips. The image shows a WS2812 ring where the LEDs are controlled with only three cables in rainbow colors.

2.1.4 Accessories - Resistors

An electrical resistor is a passive electronic component commonly integrated into circuits, particularly in the field of microcontrollers. Resistors are used to limit electric current, thereby protecting components in the circuit. For example, typical light-emitting diodes (LEDs) have a specified current of 20mA. If much more current flows in a circuit, the LED can be damaged.

Resistors can also divide electric current in a circuit. This makes it possible, for example, to read sensor values from sensors that change their electrical conductivity depending on the value being measured.

There are many different types of resistors because an individual resistance value is needed depending on the application. Selecting the appropriate resistor can be challenging, so the required resistance value is usually specified in the instructions.

2.1.5 Accessories - Sensors and Actuators

The possibilities for using sensors or actuators with the Arduino microcontroller are virtually endless. This is because the Arduino development environment is not a closed system where only prefabricated modules can be used. On the contrary, countless electronic modules from electronics stores can be used with Arduino in some way, such as reading digital or analog values. Here is an example of a (tiny) selection of typical modules that can be used in combination with Arduino microcontrollers.

(S) = sensors, (A) = Actuator, (C) = Combined module with various functions

1. Moisture (S)
2. Distance (Ultrasonic) (S)
3. Motion (S)
4. Air humidity (S)
5. Air pressure (S)
6. Current strength (S)
7. GPS receiver (S)
8. Potentiometer position (S)
9. Temperature (S)
10. Button press (S)
11. Droplet (S)
12. Slider (S)
13. Tilt (S)
14. Sound level (S)
15. Tilt / Acceleration (S)
16. Electric voltage (S)
17. Color detection (S)
18. Infrared signals (S)
19. Luminance (S)
20. GSM mobile communication (C)
21. pH value (S)
22. Magnetic fields (S)
23. Flames / Fire (S)
24. Heart rate (S)
25. RFID - Radio frequency code (C)
26. Water level / Level (S)
27. Gas (CO, alcohol, etc.) (S)
28. Fingerprint (C)
29. Impacts / Vibrations (S)
30. Microswitch (S)
31. Light barriers (S)
32. Vibrations (S)
33. UV light (S)
34. Fine dust (S)
35. Distance (Infrared) (S)

2.2 Software

The software used to program the microcontroller is open-source software and can be downloaded for free from www.arduino.cc. In this "Arduino software," you write small programs called "sketches" that the microcontroller will later execute. These sketches are then transferred to the microcontroller via USB cable. How this works is covered in the "Programming" section.

2.2.1 Installation

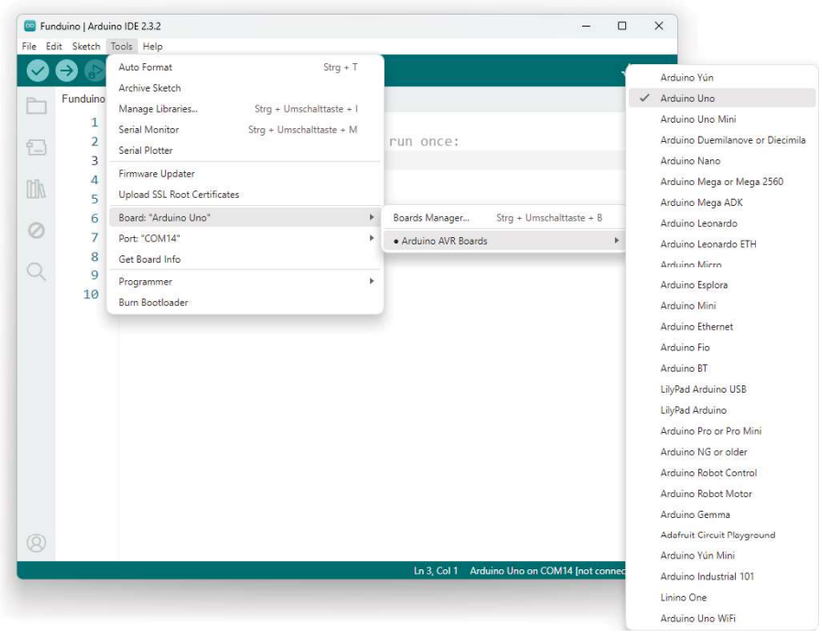
Now, the Arduino software and the USB driver for the Arduino board need to be installed one after the other.

2.2.2 Setup of the Arduino software

The latest version of the Arduino software can be downloaded from the website www.arduino.cc. After downloading the program file, the installation begins. If the installation does not start automatically, it must be started by double-clicking on the downloaded program. During this installation, no Arduino board should be connected to the computer.
After successful installation, open the Arduino software folder and start the program with the file "arduino.exe".

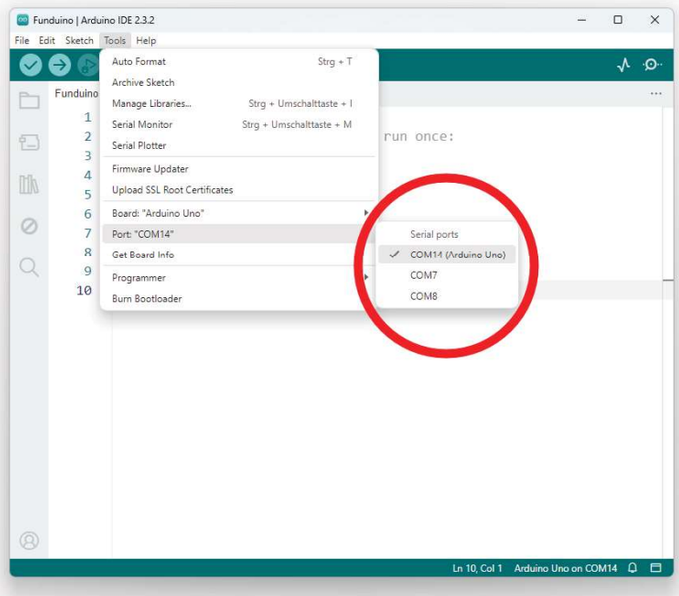
There are **two important settings** to consider in the software, which must be made in the "Tools" section.

a) The board to be connected to the computer must be selected. The "Funduino Uno" board is recognized here as "Arduino Uno," and the "Funduino MEGA2560" board is recognized as "Arduino MEGA 2560."

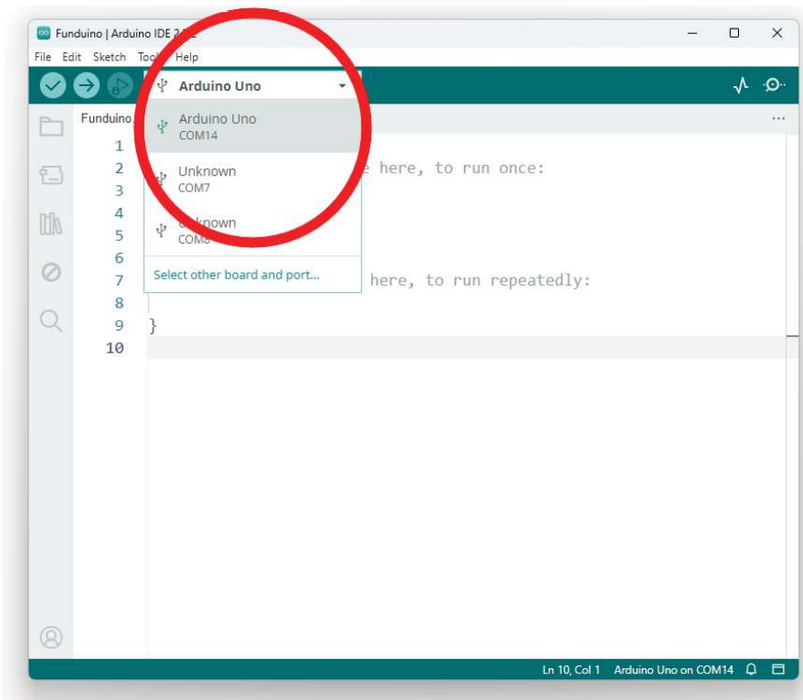


b) The correct "Serial Port" must be selected. This is important so that the PC can identify which USB port the board is connected to. However, the selection is only possible if the driver is correctly installed and the microcontroller is connected.

In software versions from version V2.0 onwards, microcontrollers and ports are often automatically detected.



If it is not clear which port belongs to the respective connected microcontroller, this can be checked with the following procedure: Without the Arduino microcontroller connected to the PC, click on 'Port' in the 'Tools' submenu in the software. There will already be one or more ports visible, such as 'COM1', 'COM4', 'COM7' ... The number of ports displayed is independent of the number of USB ports on the computer used. If the board is later correctly installed and connected, another port will be displayed here.



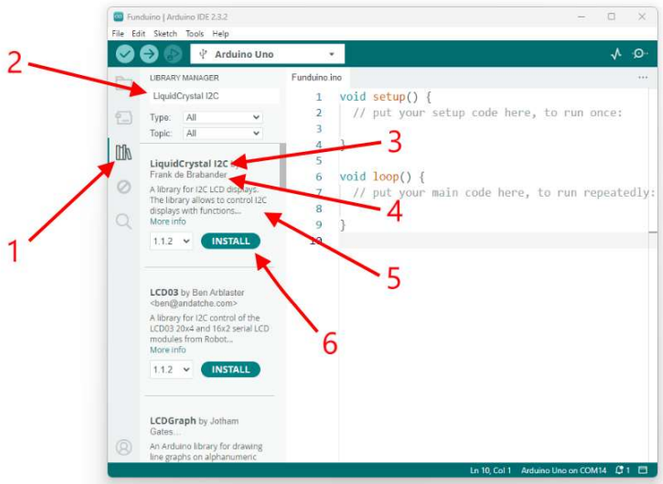
2.2.3 USB driver installation

Ideally, when installing drivers for Arduino boards or Arduino-compatible boards with original ATMEL chips (such as UNO or MEGA from Arduino or Funduino), the microcontroller board is connected to the PC and automatically installed.
However, depending on the system, the driver may not always be automatically detected and installed. In that case, you should manually select the driver during the installation process. It is located in the Arduino program folder in the "Drivers" subfolder.
Double check: In the Windows Control Panel of the computer, you can find the "*Device Manager*". After a successful installation, the Arduino board will be listed here. If the installation was not successful, either nothing special will be found here, or there will be an unknown USB device with a yellow exclamation mark. In this case, click on the unknown device in the Device Manager, select the "Update Driver" option, and then follow the on-screen instructions.

In addition to Arduino boards or Arduino-compatible boards, there are now many other microcontroller boards that are compatible with the Arduino development environment but are based on completely different microcontrollers. Therefore, these drivers are not included in the Arduino software and must be installed separately.
The most well-known chipset often used in the cheapest boards is the "**CH340**" USB chipset. This chipset is used in a variety of UNO, MEGA, NANO, and similar microcontroller boards.
To install the drivers, you need to download the appropriate drivers from the manufacturer or retailer. Usually, you need to pay attention to whether you are using Windows, Mac, or Linux. During the subsequent installation, administrative rights are typically required to ensure that the driver is installed correctly.

Practical Tip: Before working with the microcontroller, check if installing external drivers is possible on the desired computer. This is especially important in larger IT rooms with permission settings, etc., to avoid unwanted delays.

2.2.4 Adding Libraries



A library can be useful for some projects as it can simplify programming. By using a library, functions can be accessed in a sketch without having to be completely written out in the sketch. In the practical part of this guide, references to such libraries are made multiple times. These libraries need to be added to the Arduino software as needed in order to be used. There are various ways to add a library to the Arduino software. The easiest way is by clicking on the book icon (1) on the left side of the software called "Library Manager".

In the upper area (2), the desired library can be searched for and viewed using the search field. The results display the name (3) of the library, as well as the author (4). Below that is a brief description of the library (5). By clicking the "INSTALL" button (6), the library will be downloaded and installed. When installing program libraries, example sketches are simultaneously added to the Arduino software. These examples can be found under "File > Examples" and provide a good insight into the various functions of the respective library.

Alternatively, libraries can be searched for and included in the Arduino software under "Sketch > Include Library > Manage Libraries...".

There is also the option to download a library from an external site and include it using the ".ZIP Library..." function. For those who want to get a detailed overview of the functions of a library, the corresponding files can be searched for on the hard drive in the Arduino directory and then opened with an editor or with the Arduino software. A library typically consists of at least two files with the file extensions ".h" and ".cpp". The file with the ".h" extension contains or describes the functions, while the file with the ".cpp" extension contains the actual source code.

It is also possible to create your own libraries, but this is only advisable for advanced users.

2.3 Alternative Software

Besides the Arduino software, there are other ways to program Arduino boards. These are often based on programming through a graphical programming interface, which is particularly known in the education sector through "Scratch" or programming LEGO Mindstorm.

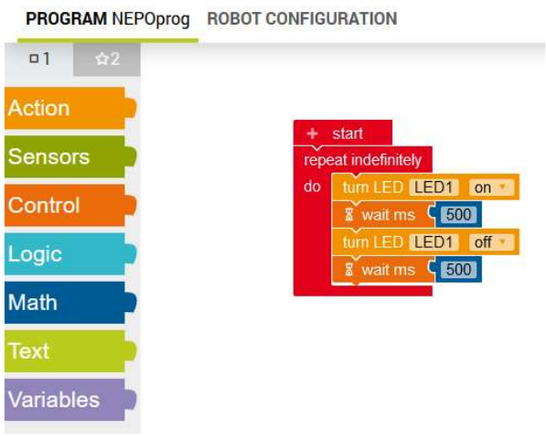
In this type of programming, all elements are essentially assembled by drag-and-drop. The elements are represented by their intuitively understandable depiction, such as programming commands through block images. This type of programming particularly motivates students, as typing code is often initially considered complicated. Another advantage is that by eliminating the "typing" of program codes, syntax errors cannot creep in. However, a disadvantage is that the "real" writing of program codes cannot be learned in this way.

2.3.1 Open Roberta

In the German-speaking region, "OpenRoberta" (https://lab.open-roberta.org) is very well-known. With OpenRoberta, not only Arduino boards, but also many other microcontrollers and robots can be programmed. It is therefore cross-hardware functional and particularly suitable for educational institutions to teach programming to students.

For programming Arduino boards, there is the "NEPO4ARDUINO" section in the Openroberta LAB. In the Openroberta LAB, an additional program is required for the USB connection between the browser and the microcontroller. The installation information is somewhat hidden. You can find it by clicking through the following menu path:

help – general help – Set Up - NEPO4ARDUINO



The image shows a program used to make an LED blink on the Arduino board. In this case, the Openroberta visual programming interface was used.

2.3.2 English

In the English-speaking world, the most well-known visual programming platforms are Mblock (http://www.mblock.cc) and S4A "Scratch for Arduino" (http://s4a.cat). S4A is popular because it is very similar to the regular "Scratch," which is already established in many schools for computer science classes.

3. Programming

To make an Arduino microcontroller do what the user demands, a small program is written using the Arduino software. This program is referred to as a "Sketch" in the Arduino development environment. Once a Sketch is completed and successfully verified in the Arduino software, it is loaded onto the memory of the microcontroller and executed immediately.

3.1 Basic structure for a Sketch

A sketch can initially be divided into three sections, as illustrated here in color.

```
int LED=9;                                                    Section 1
int brightness=0;
int x=5;

void setup()                                                  Section 2
{
 pinMode(LED, OUTPUT);
}

void loop()                                                   Section 3
{
 analogWrite(LED, brightness);
 brightness= brightness + x;
 delay(25);
 if(brightness ==0 || brightness == 255)
 {
 x = -x;
 }
}
```

3.1.1 Brightness 1 - Naming variables

In the first section, elements of the program are named. For example, variables are defined there, or so-called program libraries are loaded. This part is not mandatory for every sketch.

3.1.2 Section 2 - Setup

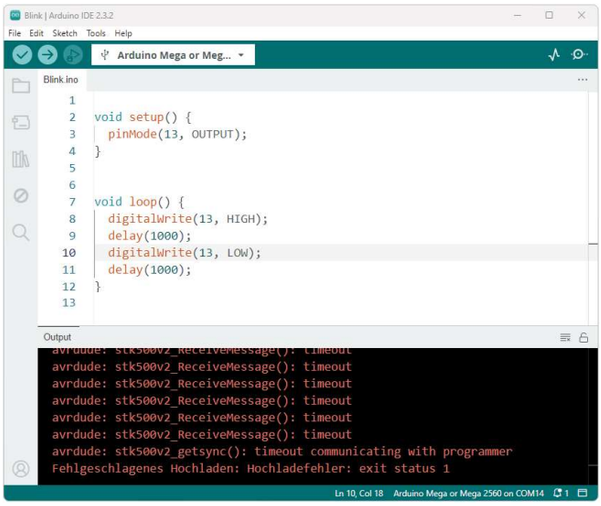
The second section is called "Setup". The Setup is executed by the board only once and is mandatory for every sketch, even if no entries are made in this section. In the Setup, for example, it is determined which pin (slot for cables) on the microcontroller board is an output or an input. Defined as an output, a voltage can be output at the respective pin (for example, to light up an LED at this pin), and defined as an input, a voltage can be read at the pin (for example, the voltage values of a sensor).

3.1.3 Section 3 - Loop

The "Loop" section is continuously repeated by the board and can therefore be referred to as the main part of the sketch. The microcontroller processes the sketch once completely until the end and then starts again at the beginning of the Loop section. Advanced users may outsource individual program sections to subroutines, which are then only called by the "Loop" and may also pass data for further processing. These outsourcings are not further discussed in this guide.

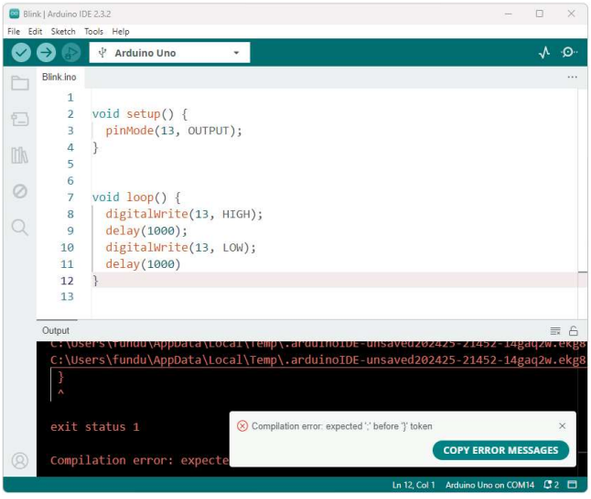
3.2 Common sources of errors

When programming microcontrollers, errors can occur at many points. The most common "beginner mistakes" when working with Arduino software are the following two:



1) The board is not installed correctly or the wrong board is selected. When uploading the sketch, an error message is displayed in the lower part of the software, which looks something like the one shown on the right. The error message then contains a note "not in sync".

2) There is an error in the sketch. For example, a word, a variable, or a command is misspelled, or a curly brace or semicolon is missing. In the example on the left, the curly brace that initiates the Loop section is missing. The error message often starts with "expected...". This means that the program is expecting something that is not yet present.



3.3 Structure of the instructions

The structure of the following instructions is very similar.

1. The instructions start with a description of the task to be completed in the guide. Additionally, if necessary, there is an explanation of the components used.

2. Before each respective sketch, there is a diagram illustrating the wiring of all modules.

3. The printed sketches in the following instructions consist simultaneously of program code and a description explaining the effect of the respective program code.

In the **left section**, the **program code** is printed in black or colored font, while in the **right section**, behind the "//" symbol in gray font, the **explanation of the program code** is provided. The explanations in gray font may be entered into the Arduino software and do not affect the execution of the sketch.

Example:

Left: Programmcode in bold font **Right:** Explanations of the program code

```
void setup()                    // Here begins the setup.
{                               // Here begins a program section.
pinMode(13, OUTPUT);            // Pin 13 should be an output.
}                               // Here ends a program section.

void loop()                     // Here begins the main program.
{                               // Program section begins.
digitalWrite(13, HIGH);         // Turn on the voltage at Pin 13. (LED on)
delay(1000);                    // Wait for 1000 milliseconds (one second).
digitalWrite(13, LOW);          // Turn off the voltage at Pin 13. (LED off)
delay(1000);                    // Wait for 1000 milliseconds (one second).
}                               // Program section ends.
```

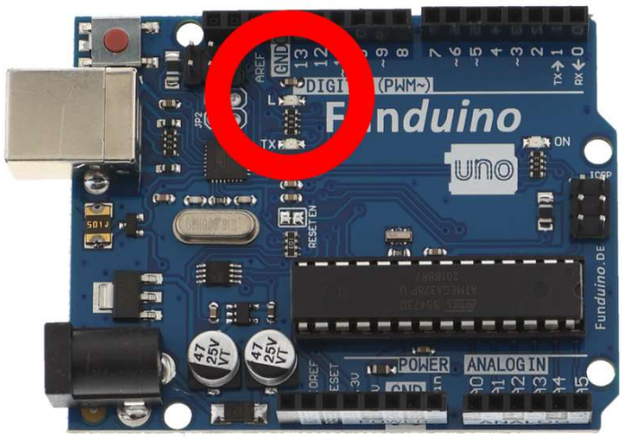
By following this system, one can quickly understand and apply the program code independently. Afterward, individuals can familiarize themselves with additional functions or modules. This guide serves as an introduction to the Arduino development environment and provides insight into programming capabilities. A comprehensive list of all program codes is provided and described on the website "www.arduino.cc" under the section "Reference".

4. Practical guides

In the following section, working with the Arduino development environment is explored through practical examples using predefined hardware.

4.1 A blinking LED

Task: A light-emitting diode (LED) should blink.



Material kit
1x Arduino-Board

On the Arduino microcontroller board, there is already an LED built into Pin 13 (for testing purposes). Often, this lamp already blinks when a new Arduino board is connected, as the blink program for testing the board may already be pre-installed depending on the manufacturer. We will now program this blinking ourselves and change the blink speed.

Circuit:
The LED already present on the board is circled in red in the image. It is only necessary to connect the Arduino board to the computer via USB cable.

Program section 1 is not needed.

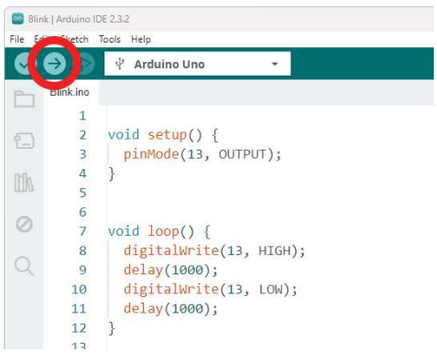
Program section 2, Setup:

For this task, we only need one pin of the microcontroller board, Pin 13. A voltage should be output at Pin 13 because the LED should light up. Therefore, in the setup, it needs to be specified that Pin 13 is an output. The explanations behind the double slash "//" in gray font may be entered into the Arduino software but do not affect the execution of the sketch. Commenting out is very useful in programming to leave information about the program for oneself or simply to jot down ideas.

We write the following sketch directly into the white input field of the Arduino software. Only the bolded program code on the left side needs to be typed. The "commented out" information about the sketch can be omitted.

```
void setup()          //Here begins the setup.
{                     //Here begins a program section.
pinMode(13, OUTPUT); //Pin 13 should be an output.
}                     //Here ends a program section.
void loop()           //Here begins the main program.
{                      //Here begins a program section.
digitalWrite(13, HIGH);//Turn on the voltage at Pin 13 (LED on).
delay(1000);           //Wait for 1000 milliseconds (one second).
digitalWrite(13, LOW);//Turn off the voltage at Pin 13 (LED off).
delay(1000);           //Wait for 1000 milliseconds (one second).
}                      //Program section ends.
```

After the last curly brace in the loop section, the sketch is completed. When the sketch is executed from within the "Loop" by the microcontroller up to the last brace, the sketch starts again from the beginning of the Loop section. In this example, this causes the LED to turn on and off repeatedly.

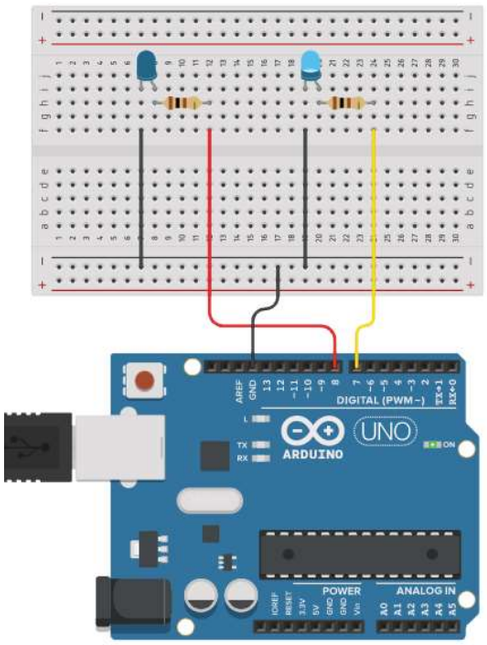


The sketch should now look exactly as shown in the image on the right. It just needs to be uploaded to the board now. This can be done using the button circled in red (top left in the software).

The program can now be varied. For example, if we want the LED to blink faster, we can shorten the delay times from 1000 milliseconds to 200 milliseconds. The new sketch must now be uploaded to the board again. With error-free input, the LED will now blink faster.

```
void setup()            //Here begins the setup
{                       //Here begins a program section.
pinMode(13, OUTPUT);    //Pin 13 should be an output.
}                       //Here ends a program section.
void loop()             //Here begins the main program.
{                       //Program section begins.
digitalWrite(13, HIGH); //Turn on the voltage at Pin 13 (LED on).
delay(200);             //Wait for 200 milliseconds.
digitalWrite(13, LOW); //Turn off the voltage at Pin 13 (LED off).
delay(200);               //Wait for 200 milliseconds.
}                         //Program section ends.
```

4.2 The alternating blinker



Task: Two light-emitting diodes (LEDs) should blink alternately.

The blink speed and rhythm can be varied.

Material kit

1x Arduino-Board
2x LEDs (blue)
2x Resistor 100 Ohm
1x Breadboard
Jumper Cables

Sketch:

```
void setup()
{                          //We start with the setup.
pinMode(7, OUTPUT);        //Pin 7 is an output.
pinMode(8, OUTPUT);        //Pin 8 is an output.
}
void loop()
{                           //The main program begins.
digitalWrite(7, HIGH);     //Turn on the LED at Pin 7.
delay(1000);                //Wait for 1000 milliseconds.
digitalWrite(7, LOW);      //Turn off the LED at Pin 7.
digitalWrite(8, HIGH);     //Turn on the LED at Pin 8.
delay(1000);                //Wait for 1000 milliseconds.
digitalWrite(8, LOW);      // Turn off the LED at Pin 8.
}                           // Here at the end, the program jumps to the start of the Loop
part. So: turn on the LED at Pin 7... etc...
```

The shorter the "delay", i.e., the pause between the alternating on and off phases of the LEDs, is chosen, the faster the blinking rhythm will be. The rhythm can be set so fast that the human eye can no longer perceive the alternating on and off phases. With this sketch, you can model the flashing sequence of emergency vehicles from various countries. Program this blue light:

Left-Pause-Left-Pause-Left-Pause-Right-Pause-Right-Pause-Right-Pause