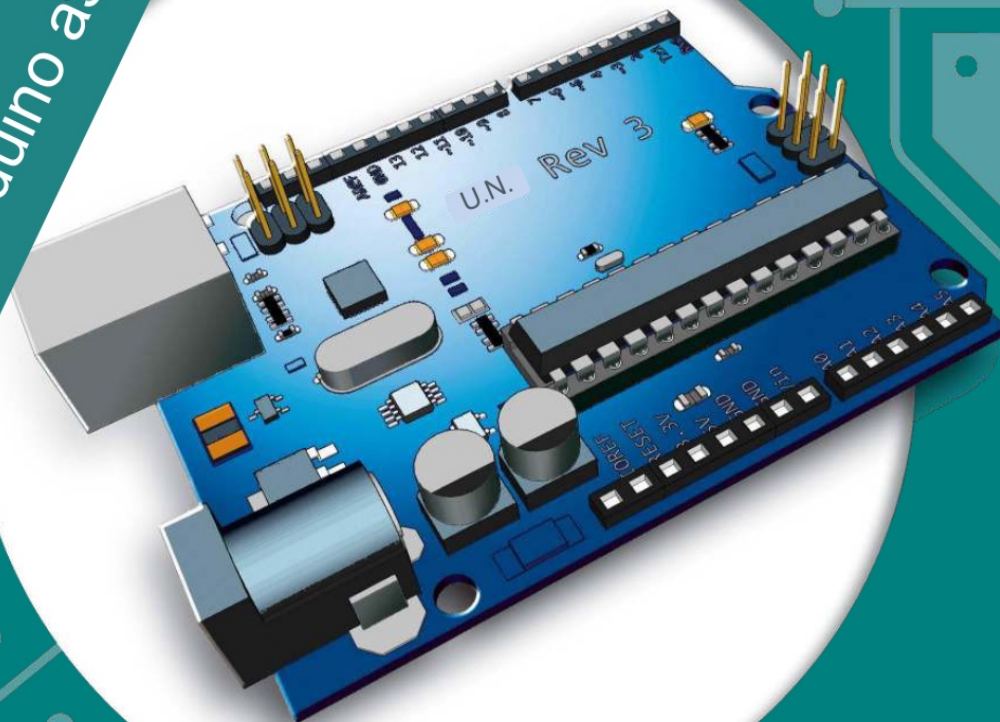


Introduction to the Arduino microcontroller

The Arduino as a control center



Version 1.0

Table of contents

| | | | |
|-----------------------------------|----|-------------------------------|----|
| 1. What is a microcontroller? | 3 | 9. LEDs dim and colors | 15 |
| 2. It lights up! | 4 | 10. If means if | 17 |
| 3. Program for programming | 6 | 11. Inputs with buttons | 18 |
| 4. The first flash | 8 | 12. Repetitions with while | 19 |
| 5. Tones, subprograms, transistor | 10 | 13. Plan programs | 20 |
| 6. Variables | 12 | 14. LCD display and libraries | 21 |
| 7. Show texts and values | 13 | 15. Control motors | 22 |
| 8. For loop | 14 | Index | 23 |

What is a Microcontroller?

1

3

Microcontrollers are small electronic components that represent the brain in automatically functioning devices, i.e. the control center that, for example...

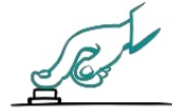


...controls the washing
program in a washing
machine and regulates
the temperatures



...in an airbag
decides whether there
is a crash and
then triggers the
ignition within 15 ms

...a sliding door
opens
automatically when people
approach



...or controls
processes in your
next project.

Microcontroller

The abbreviation μC is often used for the word microcontroller. The μ (pronounced "mü") is a Greek letter and is used as an abbreviation for "micro".

Arduino

Arduino is the name of a bar where the inventors of the Arduino often met in 2005.

This entire board is called an Arduino.

This is the actual microcontroller with its many connections, the so-called pins.

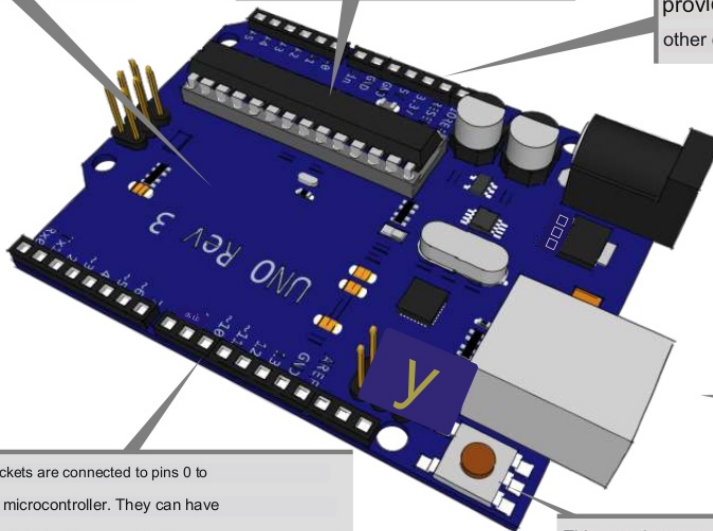
These connectors (sockets) provide voltages for connecting other components.

Here you can connect a battery or a power supply to operate the Arduino without a USB port.

You can transfer programs to the Arduino via the USB port.

These sockets are connected to pins 0 to 13 of the microcontroller. They can have two different states: they are called HIGH and LOW.

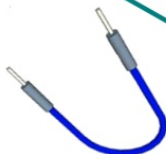
This reset button restarts the microcontroller.



It lights up!

In this chapter you will get to know the components and connections on the Arduino board, with which you can make a light-emitting diode light up:

You can initially imagine a light-emitting diode (LED) like a light bulb. However, it only conducts electricity from the longer connecting wire (called anode) to the shorter connecting wire (cathode) and breaks down very quickly if you use it without a protective resistor.



There are very practical cables for working with breadboards, the so-called jump wires. If such a cable has two plugs, it is referred to as a male-male cable, with a plug and socket

as a male-female cable and

with two sockets as

a female-female cable.

The bread

board (or

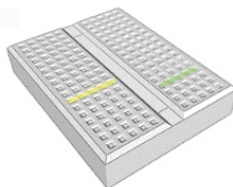
breadboard) is used to electrically

connect cables to each other without

soldering. For this purpose, 5 sockets are connected to each other in a row (see, for example, yellow or green marking). To connect two cables together,

you just have to plug

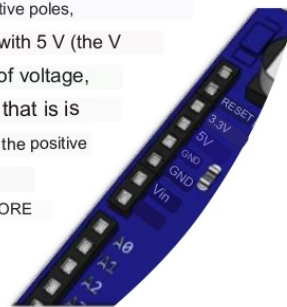
them into two sockets in the same row of five.



A resistance

is a component that can be used to protect light-emitting diodes, for example. It conducts electricity poorly - the worse, the higher its so-called resistance value (specified in the unit Ohm).

The Arduino provides various electrical potentials on this socket strip. The most important connections are three negative poles (labeled GND) and three different positive poles, one with 3.3 V, one with 5 V (the V stands for the unit of voltage, the volt) and V_{in}, that is is connected directly to the positive pole of the connected battery. CORE



Very important: When building a circuit, you should never create a direct connection between a plus and a minus pole - this is a short circuit that can destroy the Arduino and, in very clumsy cases, the connected computer.

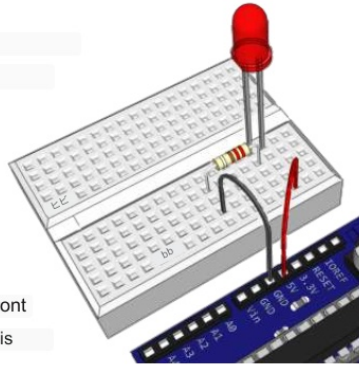
2.1

Build the circuit shown using an LED, a 220 ohm resistor (color combination red-red-brown and then silver or gold) and a few jump wires.

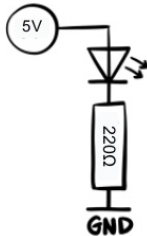
Note: In the circuit, the electricity should flow from the positive pole (5V) through the LED and the protective resistor to the negative pole (GND).

So make sure to insert the LED the right way round: Long connection to the positive pole.

It doesn't matter whether the resistor is installed in front of or behind the LED. The only important thing is that it is connected in series.



2.2



Technicians and engineers would represent the circuit as a schematic. In a circuit diagram, each component has a (usually European or even worldwide) defined circuit symbol. Create an overview page in your notebook on which you gradually list all the circuit symbols and their meaning. You can find out the first three symbols (for LED, resistor and negative pole) by comparing this circuit diagram with your circuit.

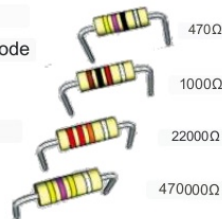
2.3

You already know coding from computer science in class 7 - the values of resistors are coded as color rings. Can you figure out the code using the examples on the right?

| | |
|--------|---|
| black | 0 |
| brown | 1 |
| red | 2 |
| orange | 3 |
| yellow | 4 |
| green | 5 |
| Blue | 6 |
| lila | 7 |
| Gray | 8 |
| white | 9 |

There are two tips: The color table is only part of the coding. And: you always start reading the code on the side where the rings are closely spaced. The fourth ring, usually silver or gold, indicates how precise the resistance is. For gold it is 5%, for silver it is 10% and 20% if the fourth so-called tolerance ring is completely missing.

What value would a resistor with the ring colors green-green-green have?

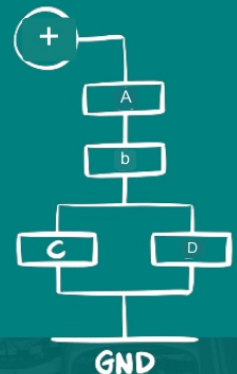


Series connection and parallel connection

Components can be arranged in a circuit as a series connection or as a parallel connection.

Two components are connected in series when electricity flows through one component and then through the other. With a parallel connection, there are multiple paths for electricity.

In the example, components A and B are connected in series, C and D are connected in parallel.



Program for programming

Where do you get the

development environment?

For home use, you can

download the development

environment from the official

website of the arduino.cc

project. It is available there

for Linux, Windows and OS X.

The Arduino software is

so-called open source

software. Anyone can

simply continue to program

open source software, distribute

it further or even

sell it further. You can also

find out more

about the

ideas behind

open source

at www.opensource.org or
the QR code.



Compile

Before the program can be

transferred to the Arduino,

it is translated from the

programming language

that is easy for humans to read
into a machine language

that is easy to read and very
compact for the

microcontroller. This process is called

"compiling".

Programs for microcontrollers are written in so-called development environments

(IDE=integrated development environment) on a computer and then transferred, for example via

a USB interface. The development environment for the many different Arduino

microcontrollers is called arduino.exe and looks like a normal program with a few special functions:

In this large field you write
the program, which is
also called a sketch on
the Arduino...

...and with this
button you can compile it
as a test (see left) and
check for typos.

By clicking on the
arrow, the program
is transferred to the
Arduino via USB.

In many programming
languages, programs
are structured in
"instruction blocks".
A block here consists
of a pair of curly
brackets (AltGr+7
or 0) and the lines
between them.

```

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here,
}

```

Information about
errors that occur are displayed
here.

Two instruction blocks are required: The
instructions in the `setup(){...}` block are executed
directly as soon as you start the Arduino. The
instructions in the `loop(){...}` block are then
executed and repeated again and again... This
is also called an "infinite loop".

Everything that appears in a line
after two slashes („/", slash) is considered
a comment line and is ignored
by the μ C. This program therefore
does not contain a single real
instruction, but only two empty blocks.

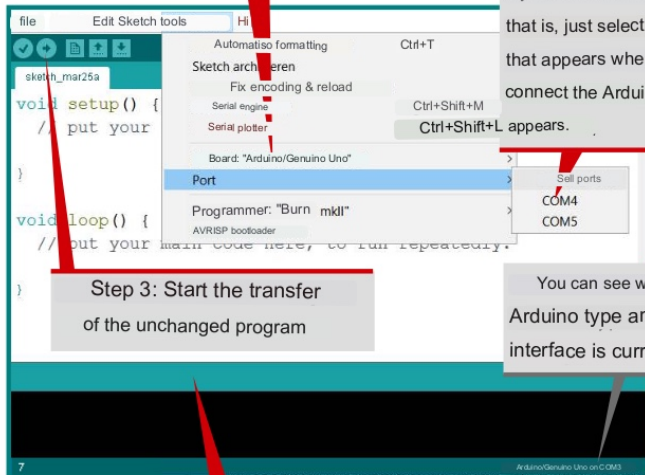
3.1

In this task you will learn how to transfer the unchanged "empty" program to the Arduino. Since the development environment is suitable for many different Arduino boards that are connected to all possible interfaces on the computer. Two settings are therefore necessary:

1. Setting: Select the correct Arduino type here. Your board is called "Arduino Uno".

2. Setting: Select the correct interface (English port).

If you don't know which one that is, just select the interface that appears when you connect the Arduino in this listing



Step 3: Start the transfer of the unchanged program

You can see which one here Arduino type and which interface is currently set.

After the successful transfer, the status line will say something like "Upload done", "Upload completed" or "Upload completed". If it didn't work, it will say "Upload failed" or "Problems uploading". Then you should change the settings check again.

3.2 Find out what the number "7" at the bottom left means. You may have a different number here. It may also be that there is no number here... then you first have to find out when it appears.

If when connecting
no interface added
comes?

Many inexpensive Arduino
Boards require one

Chinese USB drivers that
Windows and OSX do not
automatically know. You
can find it on the website of
Chinese manufacturer

http://www.wch.cn/download/CH341SER_EXE.html

download and then
start.

Error types:

There are different types
of errors that can occur
when programming:
Writing errors (so-called syntax
errors) mean that the
program cannot be
compiled into machine code.

Transmission errors mean
that the program does

not arrive on the
Arduino. Runtime errors occur

when you have made
errors in the content during
programming and the
program now does something

wrong on the

Arduino.

4

The first blink

In this chapter you will write your first program and use it to make an LED flash. To do this, the LED with its anode (longer connection) is no longer permanently connected to 5 V or 3.3 V, but to a pin of the microcontroller (in the figure it is pin 13). The program below then switches this pin alternately on (5 V) and off (0 V) - this is also called "HIGH" and "LOW". When it is "HIGH", electricity flows from the pin through the LED and the resistor to the negative terminal (GND). When it is LOW, nothing flows... and the LED is off.

This is what the program looks like:

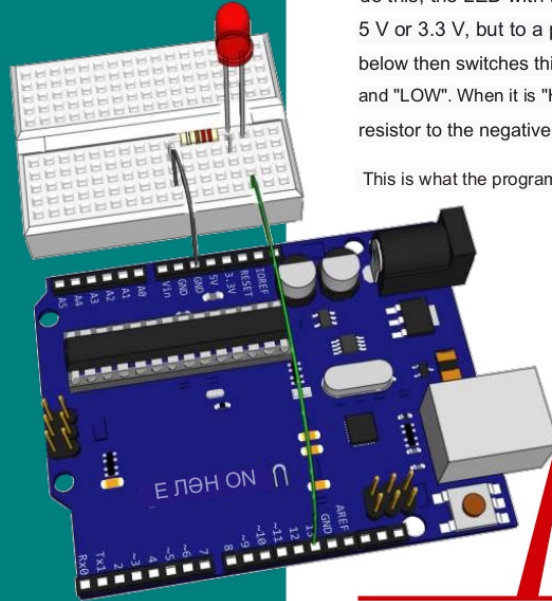
```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

So that the LED flashes, pin 13 is switched to "HIGH" and "LOW" again.

In between each time there is a wait of 1000 milliseconds.

Because these four statements are in the loop() block, they are executed over and over again.



Digital?

Digital (from the Latin digitus = finger) refers to data that can only have a few unique states, just as you can only show 10 numbers with your fingers. With digital pins there are only two states: "HIGH" and "LOW".

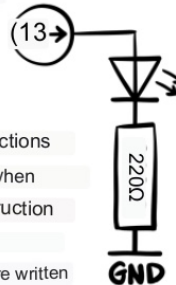
Devices that work in this way are also referred to as digital.

In order for pin 13 to be switched "HIGH" or "LOW", it must first be set to output mode (OUTPUT). The pinMode(...) instruction is used for this purpose. This only has to be done once - that's why this line is there in the subprogram "setup()".

4.1 Now connect the LED to pin 13 and transfer the program to the microcontroller.

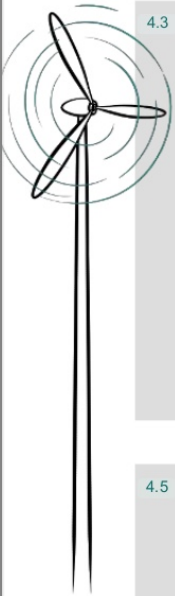
The rule of thumb is: all instructions are written in lower case, only when a new word begins in the instruction is a capital letter written.

"HIGH", "LOW" and "OUTPUT" are written completely in capital letters.



4.2 At what length of pause can your eye no longer recognize the blinking?

It is said that humans only have a recognition frequency of around 20 Hertz (Hz), so they can only recognize around 20 images per second. Does the break duration measured by match this value?



4.3

Like other tall masts, wind

turbines must have a flashing red light at their top to warn aircraft.

Since the top of the mast on a

wind turbine is not really the top

of the danger zone, its red light

flashes in a special rhythm: 1

second on, then 0.5 seconds off,

then on again for a second, but then

1.5 seconds out.

Program a continuous flashing light and

transfer the program. Is it running?

4.5

In one of these programs the LED only

flashes once, in the other it only lights

up permanently. Which is which?

Any why is this the case?

```
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
}

void loop() {
}
```

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
}
```

4.4

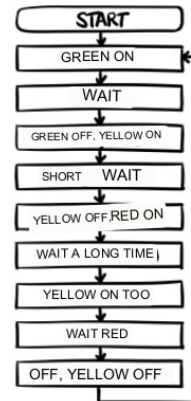
How long does the LED on pin 13 light up in this program? And how long has she been out for?

```
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
}

void loop() {
  delay(400);
  digitalWrite(13, LOW);
  delay(200);
  digitalWrite(13, HIGH);
  delay(300);
}
```

4.6

Get a green, a red and a yellow LED as well as the protective resistors for 5 V and use them to create a traffic light.



4.7

Create a list on one page of the booklet in which you can enter all 30 or so instructions and functions of the Arduino with an example. So far there are only five. Do you already know them and their exact spelling by heart?

Common error messages:

The development environment shows you where it suspects an error, partly in German and partly in English. Here are some common mistakes:

Problem uploading

Most of the time you haven't selected the right interface. Check the steps on page 7.

expected ';' before...

In German: I expect a semicolon before the line that is highlighted. Many learners forget these semicolons at the beginning.

expected '(' before...

A bracket is missing or you have used a semicolon instead of a comma.

a function definition...

Check the curly ones Brackets.

...was not declared in

The compiler encountered something unknown. You probably mistyped an instruction.

Tones, subprograms, transistor

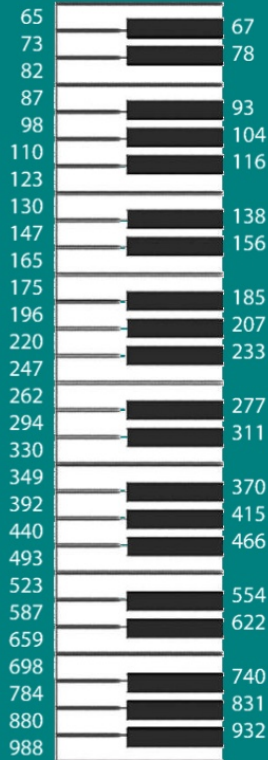
Humans can hear

sound waves with frequencies
between approximately 18

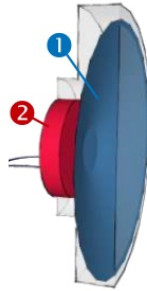
and 18,000 hertz (1 Hz is
one oscillation per second).

The following figure

shows which frequency corresponds
to which pitch:



Here you will learn how to generate sounds and play sound sequences using the Arduino and a speaker.



Many people think that a speaker immediately makes a sound when it is connected to a power source. But that is wrong: a loudspeaker consists only of an elastically suspended one

- Membrane and an electromagnet. If electricity flows through this electromagnet, the membrane is attracted; if no electricity flows, the membrane falls back again. A sound is created by quickly alternately tightening and releasing the membrane, i.e. the power supply is alternately switched on and off.

In order to produce a sound, the loudspeaker can be connected with one of its connections (no matter which one) to pin 11, for example, and the other to the negative pole. Pin 11 must be quickly switched HIGH and LOW alternately. Because you If you need to hear sounds frequently, there is an instruction that takes care of switching them on and off quickly, even if the microcontroller is doing something else:



```
void setup() {
}

void loop() {
  tone(11, 440);
  delay(1000);
  noTone(11);
  delay(3000);
}
```

This instruction ensures that pin 11 is repeatedly switched HIGH and LOW at a frequency of 440 Hz (Hertz). The Tone instruction itself ensures that pin 11 is switched to pinMode OUTPUT.

This instruction switches pin 11 permanently to LOW again. So the sound is off.

5.1

Connect a speaker to a pin and program a song. If you can't think of a better one, use "All My Ducklings":



Note: If two tones follow each other directly, you don't need a noTone instruction in between. So it also works: tone(11,262); delay(500); tone (11,294);

Parts of the program that are required multiple times (such as a chorus) do not have to be programmed multiple times. To do this, you create a so-called subprogram, which you call like a normal instruction wherever you want it to be executed.

5.2

Maybe your piece of music has a beat or a refrain that repeats itself?

Try writing this part as a subprogram: void

```

setup() {
}

void loop() {
  ...
  refrain();
  ...
  refrain();
  ...
}

void refrain() {
  tone(11,220);
  delay(300);
  ...
}

```

In the end, a subprogram feels like a self-written instruction, here for example with the self-chosen name `refrain()`. Whenever you write `refrain()`, your self-defined subprogram is called and executed.

Here you can see how the subprogram called `refrain()` is defined. It looks exactly like `setup` or `loop`, but has a self-chosen name and of course its own purpose.

How many lines can you save in your piece of music using subprograms?

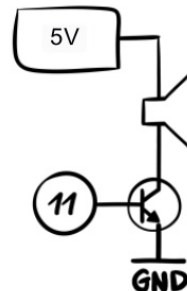
5.3

Your speaker is relatively quiet because the Arduino can only conduct low electrical power through its pins. You can get higher power from the VIN pin, which is not programmable but simply always on. With a transistor you can still use VIN for the speaker:



A transistor is like a switch that can be controlled with a pin. This control pin is connected to the base (B). If the control pin is HIGH, the switch closes and allows electricity to flow from C to E (it only works in that direction). If it is LOW, nothing flows.

Connect the transistor and speaker as shown in the circuit diagram. The sound should now become louder.



Naming: For the names of subprograms, you can use any words you choose that do not contain spaces or special characters (such as umlauts or symbols) and do not contain a number as the first character. `Grandma`, `grandpa` or `dog 16` would be possible, but `16 dog`, `bully` or a `#` is impossible. It depends on upper and lower case.

Of course, you cannot use identifiers that are already used in the programming language.

Transistor: The connections of the transistor are called base, collector and emitter. The transistor is the technological device most commonly manufactured on Earth. This is also because there are more than 1 billion transistors built into a modern computer processor alone. In the microcontroller on the Arduino there are around 100,000 - so transistors can be built very small

Naming: For

You can use any words you choose for the names of variables with the same restrictions as for subprograms.

Storage space

The Arduino has a total of around 2000 bytes of space in its memory for variables. Integers require 2 bytes each, long or float variables each require 4 bytes.

overflow

If you want to store a number that is too high in a variable, something strange happens: In an integer, for example, $32767+1$ is -32768 . This so-called overflow should be avoided.

Tip:

You can already use variables provided with a value when declared. For example, you write:

```
int a=8, b=-6;
long granny=87, p=99;
float house=3.5;
```

For decimal numbers you write a decimal point.

Variables are memory locations in which the microcontroller can remember numerical values. There are small memory locations that can only store integers from -32768 to 32767 , large memory locations that can store integers from -2147483648 to 2147483647 and memory locations that can store decimal numbers. The following example shows how to declare (name) these memory locations, how to store numbers in them, and how to reuse stored ones. Read it carefully, even though the program has no real purpose:

```
int a, b, c;
long n, m;
float x, y;
```

After the int statement you can write names of variables that are supposed to store numbers from -32768 to 32767 . These variables are also called "integers".

```
void setup() {
  a=10;
  n=100-8;
  m=2*a+15-3;
  a=200+20;
}
```

After the long statement you write the names of variables for large numbers from -2147483648 to 2147483647 .

This line makes x and y variables for decimal numbers.

The "="-sign means that the value on the right will be saved in the variable on the left. The previously saved value will be overwritten.

```
void loop() {
  tone(11, a);
  delay(n);
  noTone();
  delay(m);
}
```

You can also write invoices to the right of the equal sign. These are then calculated and the result is saved in the variable. Arithmetic symbols are $+$, $-$, $*$ and $/$.

If you save a new value in a variable, the old one is replaced.

You can use variables anywhere there are numbers.

6.1 What is the value of the variables a, n and m at the end of the example program above?

6.2 Create variables for the note durations in your piece of music (from exercise 5.2), e.g. a for eighth notes, v for quarter notes, h for half notes and use them in the delays. This way you can easily adjust the speed of your piece.

6.3

What will be the program right do well? Try it out off!

```
int f;
void setup() {
  f=100;
}

void loop() {
  tone(11, f);
  delay(100);
  f=f+1;
}
```

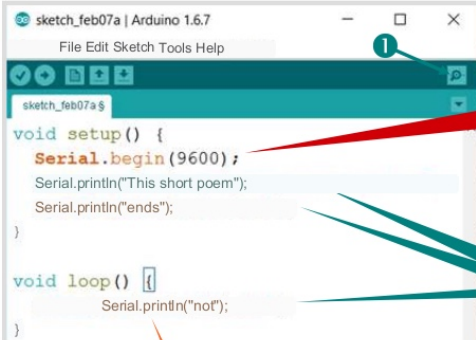
Show texts and values

7

13

This chapter is about the so-called "serial monitor". This is a program window on the computer that displays messages that the Arduino sends. You can open this program window by clicking on the magnifying glass.

This example shows you how to program the Arduino to send texts:



```
sketch_feb07a | Arduino 1.6.7
File Edit Sketch Tools Help

sketch_feb07a $

void setup() {
  Serial.begin(9600);
  Serial.println("This short poem");
  Serial.println("ends");
}

void loop() {
  Serial.println("not");
}
```

This line establishes the connection to the computer - you should only do this once in a program. The specified so-called baud rate of 9600 is explained in the marginal column.

This is how you send a message line to the computer. The message itself must be written in double quotes (Shift+2).

By the way, Println is called "Print Line" because this instruction always starts a new line after the message. There is also the Serial.print("Hello") instruction, which does not break lines.

7.1 What will the above program display in the serial monitor?

7.2 Write a program that displays a face. Maybe you can think of others? Write subprograms for "sad", "laugh"...



7.3 In the program shown on the right, the x in the Serial.print(x) statement is not in quotation marks so that it is not the letter x but the value of the variable x that is sent to the serial monitor.

What will this program display? Can you observe the variable overflow?

```
int x=0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(x);
  x=x+5;
}
```

How does it actually work? the data transmission?

You may know that data in microcontrollers and computers is stored as bit sequences, i.e. as sequences of ones and zeros, i.e. as many highs and lows. In order to send data from the Arduino to the computer, the Arduino basically switches a line HIGH and LOW depending on the data. This is called a serial transmission.

The baud rate indicates how many characters are sent or received per second. The value in Serial.begin() and bottom right of the serial monitor must match.

A notice:

The serial connection uses pins 0 and 1 of the Arduino. If possible, you should not use these pins for other tasks.

Storage space: Texts require one byte of storage space per character.

for loop

If you write statements in the subprogram `loop(){}` , they are repeated infinitely. In order to only have something executed exactly 20 or 30 times, there is the so-called for statement:

The instructions shown in blue should be repeated 30 times. For this purpose, they are combined into a block using curly brackets.

```
void setup() {
  Serial.begin(9600);
  for (int n=0; n<30; n=n+1)
  { Serial.println("Hello");
    Serial.println(n); }
}

void loop() {}
```

The for statement is written directly in front of this block. This ensures that the block is repeated and counts the repetitions. Incidentally, it is the only instruction in which the parameters are separated with a semicolon. The block is directly behind her and there is no semicolon.

Local variable

The variable that you declare in the for loop is then only known to the Arduino within the loop.

Such variables are called local variables. Variables that are declared with

an int, long or float

statement right at the beginning of the entire program and therefore apply

everywhere are called global variables.

Mathematical statements

In addition to $N < 30$, you

can also write $N \leq 30$

as a mathematical

statement in a for loop.

This means "N is less than

or equal to 30". For

counting down loops, $N > 30$

or $N \geq 30$ are also suitable.

The first parameter:

For counting, a variable is declared and its starting value is set. Here it is called n and initially has a value of 0.

The second parameter:

This is a mathematical statement. The block is only repeated as long as this statement is true. So n must be less than 30.

The third parameter

specifies what to do with n on each iteration. It says here that n is then always increased by 1.

8.1 What does the example program above do? What changes if you write $n=5$ in the first parameter? What changes if you use $n<50$ as the second parameter? What happens if the third parameter is $n=n+3$?

8.2 What would this loop do?

```
for(int f=100; f<1000; f=f+3)
{
  tone(11,f);
  delay(60);
}
```

8.3 Since 2016, the German police have also had a so-called Yelp siren with a wailing tone that rises and falls. Program a siren like this. To do this you need two for loops.

8.4 Running light: Connect five LEDs to pins 9 to 13 and program a running light: First the LED on pin 9 flashes, then the one on 10, then 11 and so on.

LEDs dim and colors

9

15

So far you can set OUTPUT pins HIGH or LOW and thus switch LEDs on or off, for example. The Arduino can also dim an LED at pins 3,5,6, 9, 10 and 11:

9.1

Connect an LED with an appropriate protective resistor to one of PINS 3, 9, 10 or 11 of the microcontroller and try changing the brightness:

```
void setup() {  
  pinMode(3, OUTPUT);  
}  
void loop() {  
  digitalWrite(3, HIGH);  
  delay(1000);  
  digitalWrite(3, LOW);  
  delay(1000);  
  analogWrite(3, 127);  
  delay(1000);  
}
```

These lines do what you already know: switching pin 3 high and low.

This line turns on pin 3 with a strength of 127. The 127 stands for 127, so 255 is "half on". At 255 the LED would be completely bright, at 0 it would be completely dark.

9.2

Write a program that slowly makes an LED

brighter and then darker again. Two for loops will help you with this.

9.3

If you use the values 127 or 128 in analogWrite(), the LED gets almost exactly half the energy. Nevertheless, it doesn't appear "half bright" to our eyes. Find out which value really looks "half bright".

Do your classmates have similar values?

9.4

On the right you see a program that writes random numbers to the serial monitor. The random generator is started once with randomSeed (analogRead(A1)), with s=random (1,7); always generates a random number from 1 to 6 and puts it into the variable s.

Can you modify the program so that you can use an LED to simulate

the random flickering of a candle?

```
int s;  
void setup() {  
  randomSeed(analogRead(A1));  
  Serial.begin(9600);  
}  
void loop {  
  s=random(1,7);  
  Serial.println(s);  
  delay(1000);  
}
```



Analog is the opposite of digital. It means that signals are processed continuously or at least in fine steps.

Pulsweitenmodulation:

By the way, the analogWrite instruction does not actually switch a pin half-way HIGH, but uses a technique called pulse width modulation (PWM): The pin is switched HIGH briefly every 2 milliseconds and then LOW again - faster than the eye can see. There are 255 steps within each 2ms interval. The more of these steps the pin is HIGH, the brighter the LED

appears.

Note: The analogWrite() instruction works ideally

with pins 9 and 10, with 3 and 11 it influences the tone() instruction, with pins 5 and 6 an LED never goes completely out. These pins are marked with the symbol on many Arduinos.

Like tone(), analogWrite() automatically switches a pin to OUTPUT.

Why red, green and blue?

In our eyes, colors are perceived with the help of the cones on the retina.

There are three types of cones:

one mainly for red, one for green and one for blue. With the three colors of an RGB LED

you can specifically stimulate the three types of cones and thus create any

color impression in the human eye.

Cell phone displays,

monitors and televisions also use

this. There - if you look

very closely - every pixel consists of light sources in exactly these three colors:

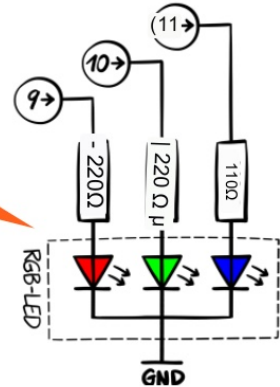


A so-called RGB LED contains three light-emitting diodes in the colors red, green and blue (hence RGB) in a housing with a common negative pole. The circuit on the right shows you how such a light-emitting diode is connected.



Everything within the dashed line is within the RGB LED. So you can see that the negative poles of all LEDs are combined to form a common negative pole (common cathode).

With RGB LEDs, the longest connection leg is usually the common negative pole. It's best to try out which of the other connections is responsible for which color (with a 220 Q protective resistor).



9.5

Find out which of the other connections is for which color and connect the LED according to the circuit diagram above.

What mixed colors result when red and green, red and blue, green and blue and all three LEDs are on?

9.7

Even more color combinations than in exercise 9.5 result if you mix the colors dimly with analogWrite(). Write a program that displays all possible color combinations, i.e. $256 \times 256 \times 256 = 16777216$ colors.

To do this you probably need three for loops one inside the other.

9.6

Look for the ten or so typos in this program.... What is the program supposed to do?

```
void setup(); {
    serialBegin(6900);
}

void loop {
    for(int i=0;i<10;i=i+1){
        Serial.print("Huhu");
        for(u=0,u<100;u=u+1){
            Serial.print(i);
            Serial.print(",");
            Serial.print(u);
        }
    }
}
```

If means if

10

17

If is one of the most important structures in most programming languages. If you write an if() in front of a statement, you can ensure that it is only executed under a certain condition. The following heads or tails program shows you how to do this:

```
int x=0;

void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(A1));
  x=random(1,7);
  if(x<6) Serial.println(x);
  if(x>5)
  { Serial.print("You have a 6!");
    Serial.print("Roll the dice again!");
  }
}

void loop() {}
```

The if(...) statement works like this:
The following statement is only executed if the mathematical statement between the round brackets is true.

To ensure that these two statements are only executed when x is greater than 5, they are combined into a block using the curly brackets.

In addition to $x>50$, $x<50$, $x\leq 50$ and $x\geq 50$ you can also write $x!=50$ (not equal) and $x==50$ (equal). Attention: This is the same "is written (unlike in mathematics class) with a double equal sign. Of course, equations are also possible with this, e.g. $x+10==y*2$.

10.1

Write a rock-paper-scissors program. Each time you press the reset button, it should randomly display one of the three words "scissors", "rock" or "paper" on the serial monitor. One possibility would



be to generate a random number. If it is 1 If the number is 2, "Scissors" is displayed; if it is 2, "Rock" is displayed; if it is 3, "Paper".

10.2

You can write an else after if. The block following else is only executed if the mathematical expression in if was not true. So what will this program do with an LED on pin 9?

```
long i=1;
void setup() {}
void loop() {
  if(i<100000) {
    analogWrite(9,100);
  } else {
    analogWrite(9,255);
  }
  i=i+1;
}
```

Why the double equal sign?

In the round brackets the

If statement you have to write $x==50$ for the mathematical statement "X is equal to 50" when programming. If you just write the simple equal sign " $X=50$ ", this means that the value 50 is stored in the variable x.

Notation

There will be several after if() If instructions are grouped together in a block, they are usually indented by a few spaces. You can also have this done automatically by pressing Ctrl+t.



Inputs with buttons

Switches and buttons?

The difference between

switches and buttons is: With

switches, the electrical

state remains intact after

you press them, with buttons

only as long as you press

them. Both buttons and

switches are collectively called

switching elements.

Switching elements with many

connections?

Many switching elements

have more than two

connections. In order to be

able to operate it on

a microcontroller, you

have to find out which of

the connections on the switch

are connected when

activated and which are not

connected when opened. A

multimeter can help you as

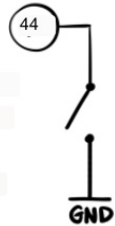
a conductivity tester.

Here you will learn how to connect switches or buttons to the Arduino

and react to them in the program when they are pressed. For this to

work, a switch must always be connected between a pin and the negative pole.

Furthermore, the pin must not be set to OUTPUT mode but to INPUT_PULLUP.



11.1 Connect a button (or switch) to pin 4 as shown in the circuit diagram.

This program displays the status of the button in the serial monitor:

```
int i;
void setup() {
  pinMode(4, INPUT_PULLUP);
  Serial.begin(9600);
}
void loop() {
  i=digitalRead(4);
  Serial.println(i);
}
```

This instruction sets the pin to INPUT_PULLUP mode. _

Here the current status of the button is queried and stored in the variable i...

...and then displayed here.

Is the following sentence correct? "If the button is closed, a 0 is displayed because pin 4 is then connected to the negative pole. If pin 4 is not connected to the negative pole because the button is not pressed or not connected at all, the result is a 1."

11.2 Write a program that turns off an LED on pin 9 when the button is not pressed and turns it on when it is pressed. If-else helps with that.

11.3 Connect two buttons and a speaker. When you press one, a low tone should be played, while the other should produce a high tone.

11.4 Write a program that counts and displays how often a button is pressed. What happens when you keep pressing the button?

11.5 The millis() instruction always

contains the number of milliseconds that the microcontroller has already been running.

Can you use this program to find out approximately how long it takes to write a character to the serial monitor?

```
long m;
void setup() {
  Serial.begin(9600);
}
void loop() {
  m=millis();
  Serial.println(m);
}
```


Repeats with while

12

19

The while statement works similarly to the if statement. However, while is not just about whether a statement or an entire block is executed, but also whether

it is repeated:

```
while (i<100) {  
  tone (10,i);  
  i=i+1;  
}
```

H

Only if the mathematical expression is true, the statement block is executed and then repeated until the expression is no longer true.

12.1

How will this program work?

Does the program ever get to the loop() subprogram?

```
int x=1;  
void setup() {  
  while (x<100) {  
    Serial.println(x);  
    x=x+1;  
  }  
  while (2<5) {  
    Serial.print("!");  
  }  
}  
void loop() {}
```

12.2

Here a button is connected to pin 4.

What do the two empty while loops

do? What does the program do?

```
void setup() {  
  Serial.begin(9600);  
}  
void loop() {  
  while (digitalRead(4)==1) {  
  }  
  Serial.print("from now");  
  while (digitalRead(4)==0) { }  
  Serial.print("until now");  
}
```

By the way, you can also link several conditions together in the mathematical expression of if() and while(): if(w<100 && u==3) means that the if block is only executed if w is less than 100 and also u is equal to 3. The two & characters must be written without spaces, as well as the two pipes here that stand for an "or": if(p<10 || p>30).... The block is executed if p is less than 10 or p is greater than 30. Wrong would be: if(p<10 || >30).

12.3 Finetuning: For this task you need two buttons and a loudspeaker. Every

time you press a button, the tone should increase by 1 Hz. Whenever you press the other button, it should decrease by 1 Hz.

Additionally, how can you use a while loop to prevent triggering more than one hertz of frequency change per key press?

Common mistake

It is a common mistake to inadvertently program a while() loop so that the mathematical expression never becomes untrue. Then the program gets stuck in this loop forever - this is called an endless loop.

Plan programs

Note: good

Flussdiagramme

Program flowcharts are flow diagrams that allow you to immediately see where you have to program a loop and where you have to program a branch. You avoid drawing arrows that go straight through each other.

Subprograms If

you have created

your own subprograms, you often draw a

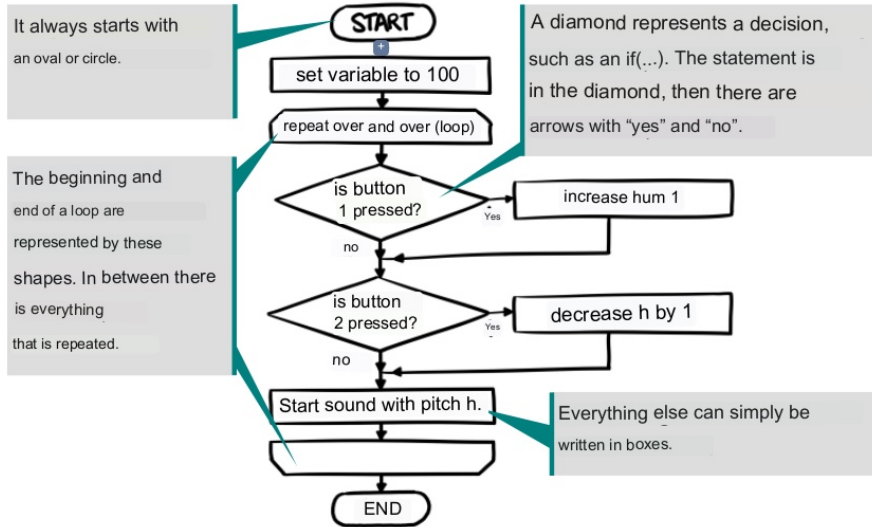
separate structure diagram for each subprogram.

That makes it clearer!

Notice

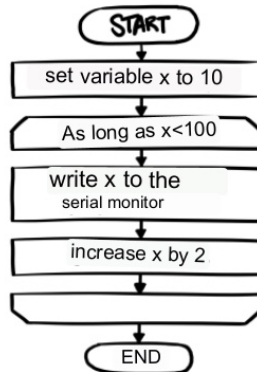
You can also draw program flowcharts on the computer using special programs (such as PapDesigner, which is free for private purposes).

For larger projects, it makes sense to think about how a program will run before you start it programs. So-called program flow charts (PAP) are used for this because they have some have a few very useful symbols, but are otherwise just plain text:



13.1

Write this program:



13.2

Design a program flow diagram

for the following sequence:
Whenever you press a button connected to pin 4, the current millis() value is written to the serial monitor.

13.3

Sketch the program flow plan for task 12.2 and task 9.7.

If you draw several loops inside each other, it increases clarity to draw the outer ones a little wider.

LCD display and libraries

14

21

Here you will learn how to operate an alphanumeric display on the Arduino in 3 steps.

The Arduino development environment does not yet provide the necessary instructions.

That's why in step 1 you'll learn how to reload an instruction library:

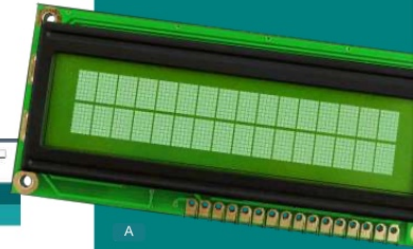
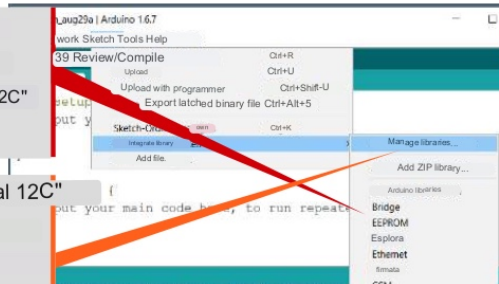
14.1

Step 1: Check in the list under

"Include Sketch Library"

whether the library "LiquidCrystal I2C"
is already in the list.

If not: Search for "LiquidCrystal I2C"
in the library loop()
management, select it, click "Install".



Alphanumeric display
is an LCD display that
can mainly display
letters and numbers. The
display shown here has
2 lines for 16 characters
each.

14.2

Step 2: Connect the add-on board to the Arduino with four cables: GND to GND,
VCC to 5V, SDA to the SDA socket and SCL to SCL.

If nothing is visible on the display,
you can adjust the contrast here.



Which parameters?

Three parameters are
required to define the display:
address, number of columns
and number of lines.

The address depends on the
placed on the back
I2C board off. The
address 0x27 matches the
boards shown, but there
are also 0x3f or 0x20.



For displays with 16
columns, the numbers 16, 2 follow.

For larger displays
with 20 columns and
4 rows, logically 20, 4.

14.3

Step 3: Try this example: What does this program do?

```
#include <Wire.h>
```

```
#include <LiquidCrystal_I2C.h>
```

```
LiquidCrystal_I2C Lcd(0x27,16,2);
```

```
void setup() {
```

```
  Lcd.init();
```

```
}
```

```
void loop() {
```

```
  Lcd.clear();
```

```
  Lcd.setCursor(3,0);
```

```
  Lcd.print("Zeit:");
```

```
  Lcd.print(millis());
```

```
  delay(100);
```

```
}
```

The first two lines integrate
two libraries into the program.

This instruction defines the display. The 3
parameters are explained to you in the marginal column.

The display starts here...

and here the ad is deleted. ...

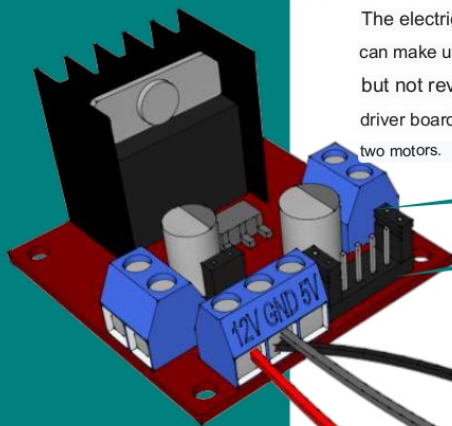
Here the cursor is placed in column 3 and line 0.

Top left corresponds to column 0 and row 0.

The quotation marks have the same effect as Serial.print();

Try: What do the two statements Lcd.backlight() and Lcd.noBacklight() do?

Control engines



IC L298

The central component on the motor driver board is called IC L298. IC stands for integrated circuit. Inside the L298 there are 8 cleverly interconnected powerful transistors built into a single housing. The circuit is also called an H-bridge.

The electrical power of the pins is not sufficient to operate a motor. With a transistor you can make use of the higher power of VIN, but you could only switch the motor on and off, but not reverse the polarity to change its direction of rotation. That's why we use a motor driver board here that can be used to control two motors.

A motor can be connected to each of these double terminals.

The motor connected to the right is controlled via the two right input pins. If one is HIGH and the other is LOW, it rotates in one direction - vice versa in the other. If both are HIGH or LOW, it stands still. The left two pins are for the left motor connection.

This plug must be connected to a GND socket on the Arduino.

The supply for the motor is connected here, i.e. a power supply or a battery pack with a voltage of 5 to 12 V.

15.1

Connect a motor to the right double terminal and connect the right two pins to pins 3 and 4 of your Arduino. Now write a program that repeatedly switches pin 3 HIGH and pin 4 LOW, waits two seconds, then switches both pins HIGH and waits again for two seconds.

If you now connect a power supply or battery pack, the motor should run, stop, run, stop...

15.2

Now connect two motors and write subprograms for "both forward", "both backward", "left forward", "right forward" and "stop".

With subprograms like this you are pretty close to controlling a vehicle that uses two engines as rear wheels:

Index

| | | | | | | | |
|-----------------------|-------|-------------------------|--------|-----------------------------|--------|--------------------------|----|
| - | 12 | dim | 15 | Lcd.clear() | 21 | Reset | 3 |
| " | 13 | Display | 21 | Lcd.init() | 21 | RGB-LED | 16 |
| #include <...> | 21 | Electromagnet | 10 | Lcd.noBacklight() | 21 | Circuit diagram | 5 |
| && | 19 | else | 17 | Lcd.print(...) | 21 | Switch | 18 |
| . | 12 | Emitter | 11 | Lcd.setCursor(...,...) | 21 | Ribbon | 14 |
| / | 6, 12 | Infinite loop | 19 | LCD-Display | 21 | Protective resistance | 4 |
| | 19 | Development environment | 6 | LED, light-emitting diode | 4 | SCL | 21 |
| ~ | 15 | expected error | 9 | Library | 21 | SDA | 21 |
| <,<=,>,>= | 14 | Farbring | 5 | LiquidCrystal_I2C | 21 | Serial.begin(...) | 13 |
| = | 12 | Mistake | 9 | local variables | 14 | Serial.print(...) | 13 |
| == | 17 | float | 12 | long | 12 | Serial.println(...) | 13 |
| a function definition | 9 | Flow Chart | 20 | loop() | 6 | serial monitor | 13 |
| alphanumeric | 21 | for(...;...;...) | 14 | LOW | 3 | setCursor(..., ...) | 21 |
| analog | 15 | curly braces | 6 | membrane | 10 | setup() | 6 |
| analogWrite(...,...) | 15 | equal sign | 12, 17 | Microcontroller | 3 | Sketch | 6 |
| quotation marks | 13 | GND | 4 | millis() | 18 | slash | 6 |
| Anode | 4 | H-bridge | 22 | Motor | 22 | Steckbrett | 4 |
| Arduino | 3 | Hertz | 10 | Motor driver | 22 | Button | 18 |
| arduino.exe | 6 | HIGH | 3 | power supply | 22 | Tolerance ring | 5 |
| Output | 8 | Hz | 10 | noBacklight() | 21 | tone(...,...) | 10 |
| backlight() | 21 | I ² C | 21 | noTone(...) | 10 | Pitches | 10 |
| Basis | 11 | IC | 22 | _or | 19 | Transistor | 11 |
| Battery pack | 22 | IDE | 6 | Open Source Software | 6 | driver | 7 |
| Baudrate | 13 | if(...) | 17 | OUTPUT | 8 | overflow | 12 |
| Conditions | 19 | INPUT_PULLUP | 18 | Oval | 20 | Transferred | 7 |
| library | 21 | int | 12 | parallel connection | 5 | and | 19 |
| Block | 6 | Integer | 12 | pinMode(...,...) | 8 | subprogram | 11 |
| Breadboard | 4 | Jump-Wire | 4 | print(...) | 13, 21 | Upload failed | 7 |
| Socket strip | 4 | cathode | 4 | println(...) | 13 | USB port | 3 |
| clear() | 21 | Piano | 10 | Upload Problems 7 Variables | 12 | | 12 |
| common cathode | 16 | collector | 11 | Program schedule | 20 | VIN | 11 |
| declaration | 12 | Comment line | 6 | Pulsweitenmodulation | 15 | void | 11 |
| delay(...) | 8 | Compile | 6 | PWM | 15 | was not declared error | 9 |
| digital | 8 | L298 | 22 | random(...,...) | 15 | while(...) | 19 |
| digitalRead(...) | 18 | speakers | 10 | randomSeed(...) | 15 | Resistance | 4 |
| digitalWrite(...,...) | 8 | Lcd.backlight() | 21 | series connection | 5 | resistance colored rings | 5 |