This tutorial kit includes everything you need for your first steps in programming: an original Arduino™ Uno, breadboard, components, a 282-page manual and software. With this kit you can build successful projects and bring to life your Arduino™.

Already done your experiment for the day?
Whether you want to build a home automation system or an LED lamp with changing colours – with the Arduino™ even beginners can successfully write their first programs and implement their very own ideas! In this tutorial kit you discover the basics of electronics and Arduino™ programming and get step-by-step instructions to put your ideas into practice.

With this tutorial kit you will perform the following projects:
- Programming with loops
- Generating random numbers
- A simple game
- Stop-watch
- Measuring voltages
- LED dimmer
- Switch-on and switch-off delay
- Music with the Arduino™
- Candlelight, courtesy of the microcontroller
- Monitoring exits
- School bell
- Keypad lock
- Voltage plotter
- Storage oscilloscope
- Temperature switch
- Romantic lights
- Timer clock
- Composing melodies
- State machines
- Capacitance meter
- ... and many more!

List of the components:
1. Arduino Uno
2. breadboard
3. 2 push-buttons
4. 1 NPN transistor BC548C
5. 1 silicon diode 1N4148
6. 1 piezo buzzer
7. 1 red LED
8. 1 green LED
9. 2 yellow LEDs
10. 3 resistors 1.5 kΩ
11. 1 resistor 4.7 kΩ
12. 1 resistor 47 kΩ
13. 1 resistor 10 kΩ
14. 1 trim potentiometer 10 kΩ PT10
15. 1 capacitor 1 μF
16. 1 insulated hookup wire ca. 1 m

In addition, you need: USB connection cable

Arduino™ is a registered trademark of Arduino LLC and the associated companies.

© 2014 Franzis Verlag GmbH, Richard-Reitzner-Allee 2, D-85540 Haar, Germany
Subject to innovation, errors and printing errors. 2014/01
ULLI SOMMER

TURN ON YOUR CREATIVITY

THE FRANZIS ARDUINO™ TUTORIAL KIT

ORIGINAL ARDUINO UNO AND 20 OTHER COMPONENTS FOR 65 PROJECTS
All circuits and programs depicted in this book are developed and tested with utmost care. Nonetheless, it is not possible to rule out all errors in the book or in the software. Publisher and author are only liable in case of intent or gross negligence according to legal regulation. Beyond that, publisher and author are only liable according to the law on product liability concerning hazards to life, body, and health and the culpable violation of essential contractual obligations. The damage claim for the violation of essential contractual obligations is limited to the contract-specific, predictable damage, unless in cases of mandatory liability according to the law on product liability.

Dear customers!
This product was developed in compliance with the applicable European directives and therefore carries the CE mark. Its authorized use is described in the instructions enclosed with it. In the event of non-conforming use or modification of the product, you will be solely responsible for complying with the applicable regulations. You should therefore take care to assemble the circuits as described in the instructions. The product may only be passed on along with the instruction and this note.

Waste electrical products should not be disposed of with household waste. Please recycle where facilities exist. Check with your local authority or retailer for recycling advice.

All rights reserved, including those of reprinting, reproduction and storage in electronical media. No part may be reproduced and distributed on paper, on storage media, or in the Internet, especially as PDF, without the publisher’s prior written permission. Any attempt may be prosecuted. Hardware and software product names, company names, and company logos mentioned in this book are generally registered trademarks and have to be considered as such. For product names, the publisher uses mainly the spelling of the manufacturer.

Arduino™ is a registered trademark of Arduino LLC and the associated companies.
# Table of Contents

## Preface

### 1 Microcontroller Basics

1.1 | Measuring  
1.2 | Controlling  
1.3 | Controlling with continuous adjustment  
1.4 | Design and mode of operation  
1.5 | Programming a Microcontroller

### 2 A Survey of Available Arduino Boards

2.1 | Arduino Mega  
2.2 | Arduino Uno  
2.3 | Arduino Leonardo  
2.4 | Arduino Ethernet  
2.5 | ArduPilot  
2.6 | LilyPad  
2.7 | USB adapter

### 3 Arduino Shields

3.1 | Arduino ProtoShield  
3.2 | Ardumoto  
3.3 | TellyMate  
3.4 | XBees radio frequency modules  
3.5 | Ethernet shield
## Components in the Tutorial Kit

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 A survey of the components</td>
<td>39</td>
</tr>
<tr>
<td>4.2 Arduino Uno</td>
<td>40</td>
</tr>
<tr>
<td>4.3 Ports and LEDs of the Arduino Uno</td>
<td>41</td>
</tr>
<tr>
<td>4.4 Power supply</td>
<td>44</td>
</tr>
<tr>
<td>4.5 Reset button</td>
<td>44</td>
</tr>
<tr>
<td>4.6 ISP port</td>
<td>44</td>
</tr>
<tr>
<td>4.7 Safety notes</td>
<td>45</td>
</tr>
</tbody>
</table>

## Use of the Components

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Jump wire</td>
<td>47</td>
</tr>
<tr>
<td>5.2 Breadboard</td>
<td>48</td>
</tr>
<tr>
<td>5.3 Push-buttons</td>
<td>49</td>
</tr>
<tr>
<td>5.4 Resistors</td>
<td>49</td>
</tr>
<tr>
<td>5.5 Capacitors</td>
<td>54</td>
</tr>
<tr>
<td>5.6 Piezo buzzer</td>
<td>56</td>
</tr>
<tr>
<td>5.7 LEDs</td>
<td>56</td>
</tr>
<tr>
<td>5.8 Diode</td>
<td>58</td>
</tr>
<tr>
<td>5.9 Transistors</td>
<td>59</td>
</tr>
</tbody>
</table>

## Installation of the Programming Environment

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Installation on Windows</td>
<td>63</td>
</tr>
<tr>
<td>6.2 Installation on Mac OS X</td>
<td>71</td>
</tr>
<tr>
<td>6.3 Installation on Linux</td>
<td>72</td>
</tr>
</tbody>
</table>

## Arduino Programming Environment

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Installation on Windows</td>
<td>63</td>
</tr>
<tr>
<td>7.2 Installation on Mac OS X</td>
<td>71</td>
</tr>
<tr>
<td>7.3 Installation on Linux</td>
<td>72</td>
</tr>
</tbody>
</table>
# Table of Contents

8 Your First Arduino Program  
8.1 | What did we do?  

9 Arduino Programming Basics  
9.1 | Bits and Bytes  
9.2 | Structure of a Program  
9.3 | Our second Arduino Program  
9.4 | Getting Started with Arduino Programming  

10 More Experiments with the Arduino  
10.1 | LED dimmer  
10.2 | Soft flasher  
10.3 | Debouncing buttons  
10.4 | A simple switch-on delay  
10.5 | A simple switch-off delay  
10.6 | LEDs  
10.7 | Switching large consumers  
10.8 | Using the PWM Pins as DAC  
10.9 | Music’s in the air  
10.10 | Romantic Candlelight, Courtesy of the Microcontroller  
10.11 | Surveillance at the Exit for Staff Members  
10.12 | An Arduino Clock  
10.13 | School Bell Program  
10.14 | Keypad Lock  
10.15 | Capacitance meter with auto-range function
10.16 | Reading potentiometers and trimmers the professional way | 239
10.17 | State Machines | 242
10.18 | 6-channel voltmeter | 247
10.19 | Programming Your Own Voltage Plotter | 250
10.20 | Arduino Storage Oscilloscope | 253
10.21 | StampPlot: a professional data logger – free of charge! | 255
10.22 | Controlling the Arduino Pins via the Arduino Ports Program | 261
10.23 | Temperature Switch | 264

11 The Fritzing Program | 268

12 The Processing Program | 270

13 Appendix | 274
13.1 | Electrical quantities | 275
13.2 | ASCII Table | 277
Preface

With many microcontroller systems, you have to work through countless data sheets that are incomprehensible for beginners. The programming interfaces are very complex and devised for professional developers with years of experience in programming microcontrollers. Thus, the access to the world of microcontrollers is unnecessarily made complicated.

The Arduino system, however, is an easily comprehensible open-source platform that is easy to understand. It is based on a microcontroller board with an Atmel AVR controller and a simple programming environment. For the human-machine interaction, you can attach a variety of analog and digital sensors that capture ambient quantities and pass the data to the microcontroller where they are processed. The program causes the creation of new analog or digital output data. There is no limit to the creativity of the developer. Whether you want to build a control system for your home or a beautiful LED lamp with changing colours: The Arduino allows even beginners from another background to write functional programs and to put their own ideas into practise.

The smooth cooperation of hardware and software is the basis for »physical computing« – the linking of the real world to the bits-and-bytes world of the microcontroller.

This tutorial kit conveys the basics of electronics and Arduino programming and shows in a plain way how to implement your own ideas.

Ulli Sommer
The CD in the Tutorial Kit

This tutorial kit contains a CD with several programs, tools, data sheets, and examples. The CD is intended to help you in working with this book. All examples in this book are contained on the CD as well.

The contents of the CD

- Arduino IDE (Integrated Development Environment)
- Sample program code
- Several tools
- Data sheets
- Circuit diagrams

GPL (General Public Licence)

You can share your own programs on the internet with other users. The sample programs are provided under the open-source GPL licence (General Public Licence). This means that you have the right to modify, publish, and share the programs according to the conditions of the GPL, provided that you make them available under the same licence terms.
System Requirements

- Windows XP (32- or 64-bit) or newer; or:
- Linux (32- or 64-bit); or:
- Mac OS X.
- CD drive
- Java

More information can be found on the following websites:

- www.arduino.cc
- www.fritzing.org
- www.processing.org

Updates and Support

The Arduino IDE is continually developed further. You can download any updates free of charges at the following website:

http://arduino.cc

Warning! Eye protection in handling LEDs

Never look directly to an LED at a short distance! This could damage your retina! This is especially true for bright LEDs in a clear housing and even more for Power LEDs. The perceived brightness of white, blue, purple, and ultraviolet LEDs gives a false impression of the real danger for your eyes. Always exercise extreme caution when using convergent lenses. Operate any LEDs according to the instructions, and never use higher currents.
TURN ON YOUR CREATIVITY

FRANZIS

ARDUINO

MORE EXPERIMENTS WITH THE ARDUINO
Now that you have worked through the fundamentals and made yourself familiar with programming the Arduino, you can start with hands-on experiments. The following projects build up on the basic knowledge you have gained in the previous chapter and extend it with new functions and programming options.

It is assumed that you already understand the program statements described so far, so you can implement the examples.

The basic mode of operation is given for all the examples, but there will be no further explanation of familiar statements. If you do not have a firm grasp on some the commands, you shall tackle them again.

In most of the following experiments, you will need the breadboard and the components included in the tutorial kit. The circuits are deliberately kept simple. You can easily follow the current flow on the breadboard without a circuit diagram.
In the previous chapter, you have become acquainted with the analog PWM output and `analogWrite`. This allows you to build a dimmer that controls the brightness of an LED. Use a red LED at analog output 3 for the next experiment. If you want to use more powerful LEDs like those by Luxeon, you have to add a transistor to the analog output in order to increase the small current of the microcontroller to the amount needed by the LED.

The example project already uses a transistor as an amplifier and shows how to use it on a digital PWM output. In this experiment, we only use the low-current LED included in the tutorial kit, but you can apply a greater load to the collector circuit like the high-power LED mentioned above or a small lightbulb for a flashlight (max. 100 mA). The push-buttons S1 (brighter) and S2 (darker) control the duty cycle of the PWM output and thus the brightness. The transistor relieves the digital pin. Only a very small current (ca. 300 times smaller than the load) flow to the base. This current is amplified by the transistor that uses the small base current to switch the larger collector current.
Figure 10.1: *Diagram of the set-up for an LED dimmer with transistor*

**Example: LED dimmer**

```cpp
// Franzis Arduino
// LED-Dimmer

int brightness=0;
int SW1=3;
int SW2=2;
int LED=11;

void setup()
{
  pinMode(SW1,INPUT);
  digitalWrite(SW1,HIGH);
  pinMode(SW2,INPUT);
  digitalWrite(SW2,HIGH);
}
```

*Time needed: 15 min*

*Difficulty: 2*
void loop()
{
  if(!digitalRead(SW1) && digitalRead(SW2))
  {
    if(brightness<255) brightness++;
    analogWrite(LED, brightness);
    delay(10);
  }
  else if(digitalRead(SW1) && !digitalRead(SW2))
  {
    if(brightness>0) brightness--;
    analogWrite(LED, brightness);
    delay(10);
  }
}

This example also demonstrates the usage of logical operators like ! and && in an if query. These comparison operations cause the push-buttons to lock each other, so that nothing happens when you press both of them simultaneously.

if(!digitalRead(SW1) && digitalRead(SW2))
{
  // statement 1
}
else if(digitalRead(SW1) && !digitalRead(SW2))
{
  // statement
}

The preceding program snippet can be verbalized as follows: If SW1 is low (0 V because the button is pressed and the pull-up resistor is active and we thus have a digi-
tal value of 0) and SW2 is high (the button is not pressed and the pull-up resistor is active, thus the digital value is 1), then execute the first block. If SW1 is high (not pressed, 5 V are applied, thus the digital value is 1) and SW2 is low (pressed, digital value is 0) then execute the code after elseif.

In short form:

If SW1 = 0 and SW2 = 1 then execute statement 1

If the first condition is not true, then test the following:

If SW1 = 1 and SW2 = 0 then execute statement 2

In the statements we increment (++) or decrement (--) the variable brightness. To avoid an overflow, the less-than query (<) and the greater-than query (>) provide an upper limit of 255 and a lower one of 0.

No matter how long you press the button, the value of the variable will never exceed 255 or drop below 0. In order to provide a more convenient way to set the brightness, a delay of 10 ms is added. Every pass takes 10 ms, which makes the adjustment of the brightness very comfortable and simple.

When you increase the delay value, the dimming process will be slower when you press a button. If you remove delay completely, the variable brightness is incremented or decremented so rapidly that you cannot observe any dimming. Instead, it looks as if the LED was turned on or off.
10.2 | Soft flasher

With the sine function, you can coax the analog output to issue a sinusoidal signal. This provides for a smooth increase and decrease in the brightness of the LED which comes in handy for some applications. This slow variation in the brightness looks as if the board had a beating heart.

The set-up is the same as in the previous example (see Fig. 10.1). The main program runs through a loop that counts from 0 to 255. The corresponding values to the numbers are retrieved from the array with the sine function table and passed as PWM values to the analog output via `analogWrite`. Using a table is significantly faster than calculating the values at run time.

```c
Example: Sine wave blinker

// Franzis Arduino

// Sine wave blinker

byte i=0;
int LED=11;

```

More lighting effects by using the sine function

Time needed: 10 min
Difficulty: 2
```cpp
void setup()
{
  // This time, we do not have to do anything
  // in here ...
}

void loop()
{
  for(i=0; i<255; i++)
  {
    analogWrite(LED, Data[i]);
    delay(5);
  }
}
```

The program uses the dynamic byte array `Data[]` whose values are assigned in the braces. It is called a dynamic array because its size is determined by the number of values defined in the braces. As we have put 256 values into the braces, the array has a size of 256 bytes. You can access the single values in the array by the index in the range from 0 to 255.

The code fetches one value (`Data[i]`) at a time and writes it to the hardware using `analogWrite`, thereby changing the duty cycle of the PWM output.

By the way: This is only a quasi-analog output. The statement is called `analogWrite`, but we only change the duty cycle of the output. Without a filter at the output,
we just get a simple PWM signal and not a true analog signal as it is issued by a real digital/analog converter (DAC). The next experiment will show how you can generate a real analog signal out of a PWM signal.

Figure 10.2: The program Sinus Tab that calculates the values of the sine wave table

The little Visual Basic .NET program Sinus Tab calculates the sine wave table that you can directly insert into your programs. You can find the program on the CD included in the tutorial kit.

If you happen to own an oscilloscope you can attach an RC circuit to the analog output (instead of the LED) and view the sinusoidal progress on the monitor of the device. An RC circuit with a 10 k\( \Omega \) resistor and a 1 \( \mu \)F capacitor will suffice.
Figure 10.3: Set-up of the RC circuit. The resistor is connected to analog output 11 on the Arduino board. The negative terminal of the capacitor is attached to ground. This circuit filters the PWM signal so that only the envelope of the sine function is visible on the oscilloscope.

Figure 10.4: The result on the oscilloscope after attaching the RC circuit
Tip More information about envelopes of signals and RC circuits can be found at:

http://en.wikipedia.org/wiki/Envelope_detector

The second example shows how you can calculate the sine function for the sine wave blinker in the program. In the previous example, we have used a table, now we will carry out the calculation directly on the microcontroller. The program is much smaller, but the calculation puts a lot of stress on the microcontroller so that the run time is significantly increased.

The \texttt{sin()} function accepts a value in radians. First, you have to convert angular degrees to radians, which is done by \(x \times (\pi / 180)\). When you multiply the result with 255 (PWM range from 0 to 255) you scale the sine function to the range from 0 to 255.

Example: \textit{Sine wave blinker with sine function}

```java
// Franzis Arduino
// Soft blinker with sine function

int ledPin = 11;
float Val;
int led;

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

Calculating the sine function in the program

Time needed: 10 min
Difficulty: 2

©

©

©

©

©

©

©

©
void loop()
{
  for (int x=0; x<180; x++)
  {
    Val = (sin(x*(3.1416/180)));
    led = int(Val*255);
    analogWrite(ledPin, led);
    delay(10);
  }
}

10.3 | Debouncing buttons

Due to their mechanical composition, push-buttons exhibit the characteristic trait of »bouncing«. Whenever you press or release the button, the signal does not change immediately to *high* or *low*, but issues short impulses that give the impression of someone rapidly operating the button.

As these impulses are too short, you cannot see this effect when you use the button to switch on or off a light bulb. However, the controller retrieves the button state so fast that he gains the impression of a button that is rapidly pressed and released. In order to determine a steady state, the button has to be debounced by means of the software.

*Figure 10.5: This is the impression the microcontroller gains at the digital input due to the bouncing effect of a button*
The problem can be avoided by retrieving the state two times with a short delay between the readings. Only when the signal level at the second reading is identical to the level at the first try, you can act on the assumption that the button was actually pressed (or depressed) and the current digital value on the input is correct. The delay should be in the range from 20 to 100 ms.

Figure 10.6: Diagram of the set-up for an LED dimmer with transistor

Required parts for the experiment

- 1 x microcontroller board Arduino Uno
- 1 x breadboard
- 1 x push-buttons
- 2 x jump wire, ca. 5 cm
Example: *Debouncing a push-button V1*

```cpp
// Franzis Arduino
// Debouncing a push-button V1

int SW1=12;

void setup()
{
    Serial.begin(9600);
    pinMode(SW1,INPUT);
    digitalWrite(SW1,HIGH);
    Serial.println("Debouncing a push-button V1");
}

void loop()
{
    if(!digitalRead(SW1))
    {
        delay(50);
        if(!digitalRead(SW1))
        {
            Serial.println("Button SW1 has been pressed");
        }
    }
}
```

In this code, text is output to Terminal when the button is pressed. The state of the button is read (query for a low state), and after a short delay (50 ms) it is read again. If it is still low, the text is printed to Terminal.

The drawback of this method is, that the programs is called so often until the button is released. Another possibility is to execute the program code and then wait for
the button to be released. The program runs through the
while(!digitalRead(SW1)); loop until the button is
pressed no longer.

Example: Debouncing a push-button V2

```
// Franzis Arduino
// Debouncing a push-button V2

byte i=0;
int SW1=12;

void setup()
{
  Serial.begin(9600);
  pinMode(SW1,INPUT);
  digitalWrite(SW1,HIGH);
  Serial.println("Debouncing a push-button V2");
}

void loop()
{
  if(!digitalRead(SW1))
  {
    delay(50);
    if(!digitalRead(SW1))
    {
      i++;
      Serial.print("Button SW1 was pressed ");
      Serial.print(i,DEC);
      Serial.println("x times");
      do{
      }while(!digitalRead(SW1));
    }
  }
}
```
The reverse behaviour can be obtained by placing the do while loop at the beginning. Now the code will be executed only after the button is released.

**Example: Debouncing a push-button V3**

```cpp
// Franzis Arduino
// Debouncing a push-button V3

byte i=0;
int SW1=12;

void setup()
{
  Serial.begin(9600);
  pinMode(SW1,INPUT);
  digitalWrite(SW1,HIGH);
  Serial.println("Debouncing a push-button V3");
}

void loop()
{
  if(!digitalRead(SW1))
  {
    delay(50);
    if(!digitalRead(SW1))
    {
      do{
        while(!digitalRead(SW1));
        i++;
        Serial.print("Button SW1 was pressed ");
        Serial.print(i,DEC);
        Serial.println("x times");
      }
    }
  }
}
```

Time needed: 10 min
Difficulty: 2
The following code provides an even better (and nearly perfect) solution. It is an amalgamation of the previous example. Furthermore, the results are not only retrieved twice but also compared. The value of `digitalRead` has to be identical at two points in a given time period in order to run the code. As a further addition, we turn on or off LED L on the Arduino board.

**Example: Debouncing a push-button V4**

```cpp
// Franzis Arduino
// Debouncing a push-button V4

byte i=0;
int SW1=12;
int LED=13;
int TOG=0;
byte value_1, value_2=0;

void setup()
{
    Serial.begin(9600);
    pinMode(SW1,INPUT);
    digitalWrite(SW1,HIGH);
    pinMode(LED,OUTPUT);
    Serial.println("Debouncing a push-button V4");
}

void loop()
{
    value_1=digitalRead(SW1);
    if(!value_1)
    {
        delay(50);
        value_2=digitalRead(SW1);
    }
    // Further code...
}
```
if(!value_2)
{
    i++;
    Serial.print("Button SW1 was pressed ");
    Serial.print(i,DEC);
    Serial.println("x times");
    if(TOG!=0)TOG=0;else TOG=1;
    digitalWrite(LED,TOG);
    do{
        while(!digitalRead(SW1));
    }
}

10.4 | A simple switch-on delay

As the name implies, a switch-on delay switches on a consumer (in our case, the LED L) with a delay after pressing the button. In our example, the delay is implemented by the `delay()` command and a counting loop. When you press the button, a flag stores the state und increments the variable `i`. When `i` exceeds the preset amount of milliseconds (in this case 3000 ms or 3 s), LED L is turned on and the program gets »trapped« in the `while(1)` loop.

As in the examples about the debouncing of push-buttons, the button is attached to digital pin 12 and ground. Now you have to press the push-button once and then release it in order to leave `do{}while(!digitalRead(SW1));`.

In querying the button state, the flag is set to 1. Now the incrementation of the variable `i` begins. When it exceeds 3000, the LED is turned on. Due to `delay(1)`, the increase of `i` by 1 only happens every millisecond.
Example: Switch-on delay

```c
// Franzis Arduino Tutorial Kit

int SW1=12;
int value_1, value_2=0;
int LED=13;
byte Flag=0;
int i=0;

void setup()
{
    pinMode(SW1,INPUT);
digitalWrite(SW1,HIGH);
    pinMode(LED,OUTPUT);
}

void loop()
{
    value_1=digitalRead(SW1);

    if(!value_1)
    {
        delay(50);
        value_2=digitalRead(SW1);
        if(!value_2)
        {
            Flag=1;
do{
            }while(!digitalRead(SW1));
        }
    }
}
```
if(Flag==1)i++;
if(i>3000)
{
  digitalWrite(LED,HIGH);
  while(1);
}
delay(1);

10.5 | A simple switch-off delay

The counterpart of the switch-on delay is the switch-off delay. With this, a consumer is turned off with a preset delay after pressing the button. The procedure is identical to that of the switch-on delay, but here the variable i is decremented instead of incremented.

Example: Switch-off delay

// Franzis Arduino
// Switch-off delay

int SW1=12;
int value_1, value_2=0;
int LED=13;
byte Flag=0;
int i=3000;

void setup()
{
  pinMode(SW1,INPUT);
  digitalWrite(SW1,HIGH);
void loop()
{
    value_1=digitalRead(SW1);
    
    if(!value_1)
    {
        delay(50);
        value_2=digitalRead(SW1);
        if(!value_2)
        {
            Flag=1;
            do{
            }while(!digitalRead(SW1));
        }
    }

    if(Flag==1)i--;
    if(i==0)
    {
        digitalWrite(LED,LOW);
        while(1);
    }
    delay(1);
}
10.6 | LEDs

In most of the previously described applications, one or more LEDs were used as output to test the software. You may have asked yourself how you have to calculate the series resistor in cases like these.

An LED is very much like a normal silicon diode, but it is operated in conducting direction (anode to the positive pole and cathode to the negative). There is a voltage drop along the LED, the amount of which depends on the colour (between 1.6 and 4 V).

The exact voltage is given in the data sheet for the LED and is called $V_{\text{forward}}$. The LED also needs some current so that it can light up. This current is called $I_{\text{forward}}$ in the data sheets. In this tutorial kit, we only use low-current LEDs with a maximum operating current of 2 mA.

An example for the calculation:

$I_{\text{forward}} = 2$ mA (low-current LED)
$V_{\text{forward}} = 2.2$ V

Operating voltage of the Arduino Vcc = 5 V
$R = x \Omega$ (the quantity we want to determine)

$$R = \frac{V_{\text{cc}} - V_{\text{forward}}}{I_{\text{forward}}} = \frac{5V - 2.2V}{2mA} = 1400 \Omega$$

Use a series resistor of the E12 series with a little higher value, namely 1.5 k$\Omega$, to make sure the LED will not be damaged.
As a hands-on example, we will build an LED double flasher: The LEDs attached to the digital pins 10 and 11 blink alternately three times each. This simulates the light effect of the beacon light on an ambulance car.

**Figure 10.7: Diagram of the circuit**

- 1 x microcontroller board Arduino Uno
- 1 x breadboard
- 2 x red LED
- 2 x 1.5 kΩ resistors
- 4 x jump wire, ca. 10 cm
Example: Double flasher

```cpp
// Franzis Arduino
// Double flasher

int LED_1=10;
int LED_2=11;
int i=0;
int TOG=0;

void setup()
{
    pinMode(LED_1,OUTPUT);
    pinMode(LED_2,OUTPUT);
}

void loop()
{
    for(i=0;i<3;i++)
    {
        if(TOG==0)TOG=HIGH;else TOG=LOW;
        digitalWrite(LED_1,TOG);
        delay(40);
    }
    TOG=0
digitalWrite(LED_1,LOW);
    delay(100);
    for(i=0;i<3;i++)
    {
        if(TOG==0)TOG=HIGH;else TOG=LOW;
        digitalWrite(LED_2,TOG);
        delay(40);
    }
digitalWrite(LED_2,LOW);
delay(100);
```

Time needed: 10 min
Difficulty: 3

©
The program runs through the first `for` loop and lets the LED at digital pin 10 blink three times. Then it enters the second `for` loop and causes the second LED to blink three times. After that, it waits for 500 ms and starts again.

### 10.7 | Switching large consumers

If you need more current than our port can provide (max. ±40 mA), you will have to amplify it by a transistor as you did in the dimmer project. Let us have a more detailed look at transistors and their properties.

In a transistor, a small current ($I_B$) flows to the base and provides for a larger collector current ($I_C$). The amplification (expressed as the so called $h_{FE}$ value) of small-signal transistors amounts to a factor of 100 to 1000, depending on the model. The transistor BC548C that we use in our experiments has an average amplification factor of about 300. A base current of 0.1 mA will therefore result in a collector current of 30 mA. The collector current for our transistor must not exceed 100 mA. Again, we will use an LED with a series resistor for demonstration purposes.
Figure 10.8: A transistor at the digital output of the Arduino microcontroller (μC); $I_B = \text{base current}$, $I_C = \text{collector current}$. Both base and collector current flow through the emitter. The tutorial kit contains low-current LEDs ($I_{\text{forward}} = 2 \text{ mA}$), hence the large 1.5 kΩ series resistor $R_2$.

For the $R_1$ resistor, we choose a value between 1 and 10 kΩ, depending on the application. With a BC548C transistor a 10 kΩ resistor is sufficient to fully illuminate an LED.

The resistor $R_3$ serves to protect the base against interference. When you switch on the Arduino, the digital pins have a high resistance because they are initialized as inputs. The base would be »up in the air«. To avoid this, we attach a 220 to 470 kΩ resistor directly to the base and against ground. This makes sure that the transistor connects through only when a larger current flows to the base.
The more current the consumer needs, the more current must flow to the base so that a larger collector current is possible.

The collector current is calculated as follows:

\[ I_C = I_B \times h_{FE} \] (transistor amplification factor)

The following circuit diagram shows a transistor controlling a small relay. The resistance value of resistor R5 may be 1 to 22 k\(\Omega\), depending on the coil current. In general you can use a 4.7 k\(\Omega\) resistor because the transistor works as a simple switch.

There is no interference-suppression resistor in this circuit, but you can add one in case you experience any problems. As in the previous example, you can use a 220 k\(\Omega\) resistor between base and ground.
Button S1 uses R4 as an external pull-up resistor, where R4 should have a resistance between 10 and 22 kΩ. Diode D1 prevents the inductive voltage in the relay coil from damaging the transistor when switching off. The inductive voltage is polarised in the opposite direction of the source. Thus the diode has to be inserted in a way that short-circuits the inductive voltage. In this example, the relay turns on lamp La1 when the digital pin is high (5 V).

Figure 10.10: Relay at the digital pin of the Arduino

There are many different models of relays. They all have potential-free contacts, i.e., the contact has no connection to the microcontroller circuit whatsoever.
...ABBREVIATIONS, QUANTITIES, AND UNITS
On the following pages, you will find some useful tables for abbreviations, electrical quantities, units of measurements and symbols.

13.1 | Electrical quantities

You have to differentiate between quantities like voltage, current, and resistance and the units of measurement for these quantities (volt, ampere, and ohm). Every quantity and every unit of measurement has its own abbreviation that is used in formulas. This provides for a neat and clear notation. For instance, you simply write »I = 1 A« instead of »current = 1 ampere«.
In this book, the following abbreviations are used:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Abbreviation</th>
<th>Unit</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>V or U</td>
<td>Volt</td>
<td>V</td>
</tr>
<tr>
<td>Current</td>
<td>I</td>
<td>Ampere</td>
<td>A</td>
</tr>
<tr>
<td>Resistance</td>
<td>R</td>
<td>Ohm</td>
<td>Ω</td>
</tr>
<tr>
<td>Power</td>
<td>P</td>
<td>Watt</td>
<td>W</td>
</tr>
<tr>
<td>Frequency</td>
<td>f</td>
<td>Hertz</td>
<td>Hz</td>
</tr>
<tr>
<td>Time</td>
<td>t</td>
<td>Second</td>
<td>s</td>
</tr>
<tr>
<td>Wave length</td>
<td>λ (lambda)</td>
<td>Meter</td>
<td>m</td>
</tr>
<tr>
<td>Inductance</td>
<td>L</td>
<td>Henry</td>
<td>H</td>
</tr>
<tr>
<td>Capacitance</td>
<td>C</td>
<td>Farad</td>
<td>F</td>
</tr>
<tr>
<td>Area</td>
<td>A</td>
<td>Square meter</td>
<td>m²</td>
</tr>
</tbody>
</table>
### 13.2 ASCII Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Binary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>000</td>
<td>00</td>
<td>00000000</td>
<td>Null character</td>
</tr>
<tr>
<td>SOH</td>
<td>001</td>
<td>001</td>
<td>00000001</td>
<td>Start of header</td>
</tr>
<tr>
<td>STX</td>
<td>002</td>
<td>002</td>
<td>00000010</td>
<td>Start of text</td>
</tr>
<tr>
<td>ETX</td>
<td>003</td>
<td>003</td>
<td>00000011</td>
<td>End of text</td>
</tr>
<tr>
<td>EOT</td>
<td>004</td>
<td>004</td>
<td>00000100</td>
<td>End of transmission</td>
</tr>
<tr>
<td>ENQ</td>
<td>005</td>
<td>005</td>
<td>00000101</td>
<td>Enquiry</td>
</tr>
<tr>
<td>ACK</td>
<td>006</td>
<td>006</td>
<td>00000110</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>BEL</td>
<td>007</td>
<td>007</td>
<td>00000111</td>
<td>Bell</td>
</tr>
<tr>
<td>BS</td>
<td>008</td>
<td>008</td>
<td>00001000</td>
<td>Backspace</td>
</tr>
<tr>
<td>HAT</td>
<td>009</td>
<td>009</td>
<td>00001001</td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>LF</td>
<td>010</td>
<td>00A</td>
<td>00001010</td>
<td>Line feed</td>
</tr>
<tr>
<td>VT</td>
<td>011</td>
<td>00B</td>
<td>00001011</td>
<td>Vertical tab</td>
</tr>
<tr>
<td>FF</td>
<td>012</td>
<td>00C</td>
<td>00001100</td>
<td>Form feed</td>
</tr>
<tr>
<td>CR</td>
<td>013</td>
<td>00D</td>
<td>00001101</td>
<td>Carriage return</td>
</tr>
<tr>
<td>SO</td>
<td>014</td>
<td>00E</td>
<td>00001110</td>
<td>Shift out</td>
</tr>
<tr>
<td>SI</td>
<td>015</td>
<td>00F</td>
<td>00001111</td>
<td>Shift in</td>
</tr>
<tr>
<td>DLE</td>
<td>016</td>
<td>010</td>
<td>00010000</td>
<td>Data link escape</td>
</tr>
<tr>
<td>DC1</td>
<td>017</td>
<td>011</td>
<td>00010001</td>
<td>Device control 1</td>
</tr>
<tr>
<td>DC2</td>
<td>018</td>
<td>012</td>
<td>00010010</td>
<td>Device control 2</td>
</tr>
<tr>
<td>DC3</td>
<td>019</td>
<td>013</td>
<td>00010011</td>
<td>Device control 3</td>
</tr>
<tr>
<td>DC4</td>
<td>020</td>
<td>014</td>
<td>00010100</td>
<td>Device control 4</td>
</tr>
<tr>
<td>NAK</td>
<td>021</td>
<td>015</td>
<td>00010101</td>
<td>Negative acknowledgment</td>
</tr>
<tr>
<td>SYN</td>
<td>022</td>
<td>016</td>
<td>00010110</td>
<td>Synchronous idle</td>
</tr>
<tr>
<td>ETB</td>
<td>023</td>
<td>017</td>
<td>00010111</td>
<td>End of transmission block</td>
</tr>
<tr>
<td>CAN</td>
<td>024</td>
<td>018</td>
<td>00011000</td>
<td>Cancel</td>
</tr>
<tr>
<td>Symbol</td>
<td>Decimal</td>
<td>Hexadecimal</td>
<td>Binary</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>EM</td>
<td>025</td>
<td>019</td>
<td>00011001</td>
<td>End of medium</td>
</tr>
<tr>
<td>SUB</td>
<td>026</td>
<td>01A</td>
<td>00011010</td>
<td>Substitute</td>
</tr>
<tr>
<td>ESC</td>
<td>027</td>
<td>01B</td>
<td>00011011</td>
<td>Escape</td>
</tr>
<tr>
<td>FS</td>
<td>028</td>
<td>01C</td>
<td>00011100</td>
<td>File separator</td>
</tr>
<tr>
<td>GS</td>
<td>029</td>
<td>01D</td>
<td>00011101</td>
<td>Group separator</td>
</tr>
<tr>
<td>RS</td>
<td>030</td>
<td>01E</td>
<td>00011110</td>
<td>Request to send, record separator</td>
</tr>
<tr>
<td>US</td>
<td>031</td>
<td>01F</td>
<td>00011111</td>
<td>Unit separator</td>
</tr>
<tr>
<td>SP</td>
<td>032</td>
<td>020</td>
<td>00100000</td>
<td>Space</td>
</tr>
<tr>
<td>!</td>
<td>033</td>
<td>021</td>
<td>00100001</td>
<td>Exclamation mark</td>
</tr>
<tr>
<td>&quot;</td>
<td>034</td>
<td>022</td>
<td>00100010</td>
<td>Double quote</td>
</tr>
<tr>
<td>#</td>
<td>035</td>
<td>023</td>
<td>00100011</td>
<td>Number sign</td>
</tr>
<tr>
<td>$</td>
<td>036</td>
<td>024</td>
<td>00100100</td>
<td>Dollar sign</td>
</tr>
<tr>
<td>%</td>
<td>037</td>
<td>025</td>
<td>00100101</td>
<td>Percent</td>
</tr>
<tr>
<td>&amp;</td>
<td>038</td>
<td>026</td>
<td>00100110</td>
<td>Ampersand</td>
</tr>
<tr>
<td>'</td>
<td>039</td>
<td>027</td>
<td>00100111</td>
<td>Single quote</td>
</tr>
<tr>
<td>(</td>
<td>040</td>
<td>028</td>
<td>00101000</td>
<td>Left opening parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>041</td>
<td>029</td>
<td>00101001</td>
<td>Right closing parenthesis</td>
</tr>
<tr>
<td>*</td>
<td>042</td>
<td>02A</td>
<td>00101010</td>
<td>Asterisk</td>
</tr>
<tr>
<td>+</td>
<td>043</td>
<td>02B</td>
<td>00101011</td>
<td>Plus</td>
</tr>
<tr>
<td>,</td>
<td>044</td>
<td>02C</td>
<td>00101100</td>
<td>Comma</td>
</tr>
<tr>
<td>-</td>
<td>045</td>
<td>02D</td>
<td>00101101</td>
<td>Minus or dash</td>
</tr>
<tr>
<td>.</td>
<td>046</td>
<td>02E</td>
<td>00101110</td>
<td>Dot</td>
</tr>
<tr>
<td>CHAR</td>
<td>DEC</td>
<td>HEX</td>
<td>BIN</td>
<td>Description</td>
</tr>
<tr>
<td>/</td>
<td>047</td>
<td>02F</td>
<td>00101111</td>
<td>Forward slash</td>
</tr>
<tr>
<td>0</td>
<td>048</td>
<td>030</td>
<td>00110000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>049</td>
<td>031</td>
<td>00110001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>050</td>
<td>032</td>
<td>00110010</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Decimal</td>
<td>Hexadecimal</td>
<td>Binary</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>3</td>
<td>051</td>
<td>033</td>
<td>00110011</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>052</td>
<td>034</td>
<td>00110100</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>053</td>
<td>035</td>
<td>00110101</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>054</td>
<td>036</td>
<td>00110110</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>055</td>
<td>037</td>
<td>00110111</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>056</td>
<td>038</td>
<td>00111000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>057</td>
<td>039</td>
<td>00111001</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>058</td>
<td>03A</td>
<td>00111010</td>
<td>Colon</td>
</tr>
<tr>
<td>;</td>
<td>059</td>
<td>03B</td>
<td>00111011</td>
<td>Semicolon</td>
</tr>
<tr>
<td>&lt;</td>
<td>060</td>
<td>03C</td>
<td>00111100</td>
<td>Less than</td>
</tr>
<tr>
<td>=</td>
<td>061</td>
<td>03D</td>
<td>00111101</td>
<td>Equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>062</td>
<td>03E</td>
<td>00111110</td>
<td>Greater than</td>
</tr>
<tr>
<td>?</td>
<td>063</td>
<td>03F</td>
<td>00111111</td>
<td>Question mark</td>
</tr>
<tr>
<td>@</td>
<td>064</td>
<td>040</td>
<td>01000000</td>
<td>At symbol</td>
</tr>
<tr>
<td>A</td>
<td>065</td>
<td>041</td>
<td>01000001</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>066</td>
<td>042</td>
<td>01000010</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>067</td>
<td>043</td>
<td>01000011</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>068</td>
<td>044</td>
<td>01000100</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>069</td>
<td>045</td>
<td>01000101</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>070</td>
<td>046</td>
<td>01000110</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>071</td>
<td>047</td>
<td>01000111</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>072</td>
<td>048</td>
<td>01001000</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>073</td>
<td>049</td>
<td>01001001</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>074</td>
<td>04A</td>
<td>01001010</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>075</td>
<td>04B</td>
<td>01001011</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>076</td>
<td>04C</td>
<td>01001100</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>077</td>
<td>04D</td>
<td>01001101</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>078</td>
<td>04E</td>
<td>01001110</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Decimal</td>
<td>Hexadecimal</td>
<td>Binary</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>O</td>
<td>079</td>
<td>04F</td>
<td>01001111</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>080</td>
<td>050</td>
<td>01010000</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>081</td>
<td>051</td>
<td>01010001</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>082</td>
<td>052</td>
<td>01010010</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>083</td>
<td>053</td>
<td>01010011</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>084</td>
<td>054</td>
<td>01010100</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>085</td>
<td>055</td>
<td>01010101</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>086</td>
<td>056</td>
<td>01010110</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>087</td>
<td>057</td>
<td>01010111</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>088</td>
<td>058</td>
<td>01011000</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>089</td>
<td>059</td>
<td>01011001</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>090</td>
<td>05A</td>
<td>01011010</td>
<td>Left opening bracket</td>
</tr>
<tr>
<td>[</td>
<td>091</td>
<td>05B</td>
<td>01011011</td>
<td>Backslash</td>
</tr>
<tr>
<td>\</td>
<td>092</td>
<td>05C</td>
<td>01011100</td>
<td></td>
</tr>
<tr>
<td>]</td>
<td>093</td>
<td>05D</td>
<td>01011101</td>
<td>Right closing bracket</td>
</tr>
<tr>
<td>^</td>
<td>094</td>
<td>05E</td>
<td>01011110</td>
<td>Caret</td>
</tr>
<tr>
<td>_</td>
<td>095</td>
<td>05F</td>
<td>01011111</td>
<td>Underscore</td>
</tr>
<tr>
<td>`</td>
<td>096</td>
<td>060</td>
<td>01100000</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>097</td>
<td>061</td>
<td>01100001</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>098</td>
<td>062</td>
<td>01100010</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>099</td>
<td>063</td>
<td>01100011</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>100</td>
<td>064</td>
<td>01100100</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>065</td>
<td>01100101</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>102</td>
<td>066</td>
<td>01100110</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>103</td>
<td>067</td>
<td>01100111</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Decimal</td>
<td>Hexadecimal</td>
<td>Binary</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
<td>--------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>h</td>
<td>104</td>
<td>068</td>
<td>01101000</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>105</td>
<td>069</td>
<td>01101001</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>106</td>
<td>06A</td>
<td>01101010</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>107</td>
<td>06B</td>
<td>01101011</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>108</td>
<td>06C</td>
<td>01101100</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>109</td>
<td>06D</td>
<td>01101101</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>110</td>
<td>06E</td>
<td>01101110</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>06F</td>
<td>01101111</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>070</td>
<td>01110000</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>113</td>
<td>071</td>
<td>01110001</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>114</td>
<td>072</td>
<td>01110010</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>115</td>
<td>073</td>
<td>01110011</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>116</td>
<td>074</td>
<td>01110100</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>117</td>
<td>075</td>
<td>01110101</td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>118</td>
<td>076</td>
<td>01110110</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>119</td>
<td>077</td>
<td>01110111</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>120</td>
<td>078</td>
<td>01111000</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>121</td>
<td>079</td>
<td>01111001</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>122</td>
<td>07A</td>
<td>01111010</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>123</td>
<td>07B</td>
<td>01111011</td>
<td>Left opening brace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>124</td>
<td>07C</td>
<td>01111100</td>
</tr>
<tr>
<td>}</td>
<td>125</td>
<td>07D</td>
<td>01111101</td>
<td>Right closing brace</td>
</tr>
<tr>
<td>~</td>
<td>126</td>
<td>07E</td>
<td>01111110</td>
<td>Tilde</td>
</tr>
<tr>
<td>DEL</td>
<td>127</td>
<td>07F</td>
<td>01111111</td>
<td>Delete</td>
</tr>
</tbody>
</table>
This tutorial kit includes everything you need for your first steps in programming: an original Arduino™ Uno, breadboard, components, a 282-page manual and software. With this kit you can build successful projects and bring to life your Arduino™.

**LIST OF THE COMPONENTS:**
1. Arduino Uno
2. breadboard
3. push-buttons
4. NPN transistor BC548C
5. silicon diode 1N4148
6. piezo buzzer
7. red LED
8. green LED
9. yellow LEDs
10. 3 resistors 1.5 kΩ
11. 1 resistor 4.7 kΩ
12. 1 resistor 10 kΩ
13. trim potentiometer 10 kΩ PT10
14. capacitor 1 μF
15. insulated hookup wire ca. 1 m

In addition, you need: USB connection cable

Arduino™ is a registered trademark of Arduino LLC and the associated companies.

© 2014 Franzis Verlag GmbH, Richard-Reitzner-Allee 2, D-85540 Haar, Germany

Subject to innovation, errors and printing errors. 2014/01