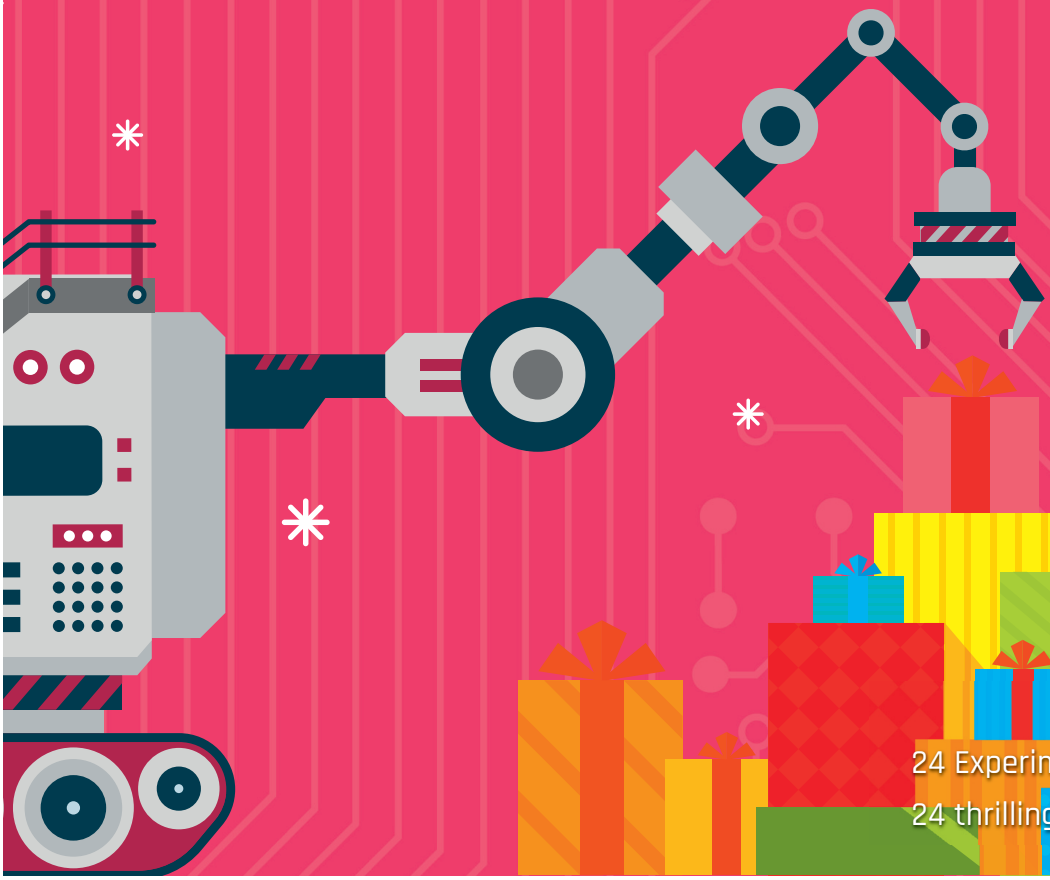
MAKER FACTORY

ADVENTSKALENDER FÜR RASPBERRY PI

ADVENT CALENDAR FOR RASPBERRY PI

COMPATIBLE WITH RASPBERRY PI



24 Experimente, die Spaß machen

24 thrilling experiments

Overview of all experiments

Raspberry Pi Advent Calendar 2019

This Advent calendar contains a hardware experiment with the Raspberry Pi for every day. The experiments are programmed using Scratch and Python. Both programming languages are pre-installed on the Raspberry Pi. All experiments work with Raspberry Pi 3, Raspberry Pi 3 B+ and Raspberry Pi 4 with the current version of the Raspbian operating system.

Controlling simple electronics using a normal PC or even a notebook - even if only for a few LEDs - can be quite a challenge for any hobby programmer. A PC simply lacks the necessary interfaces for this. In addition, the Windows operating system is unthinkably unsuitable for communicating with electronics.

The Raspberry Pi is – although it doesn't look like it at first – is a full-fledged computer. Many things work a little slower than you are used to from modern PCs, but the Raspberry Pi is also much smaller and cheaper than a PC.

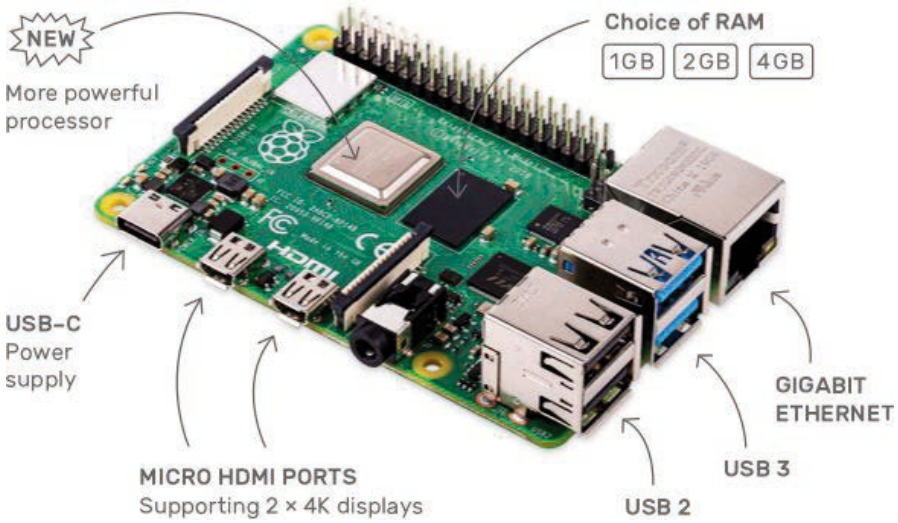
Programs to download
The programs used in the Advent calendar can be downloaded here: **www.buch.cd**. Enter the code 15045-5 **for this product in the input field**.

1. Day

1st day

Today in the Advent calendar

- 1 pinboard
- 1 LED red
- 1 resistor 220 ohms
- 2 GPIO connection cable



The Connections on the Raspberry Pi (Graphics: Raspberry Pi Foundation - Creative Commons Licence)

Preparing the Raspberry Pi

To put the Raspberry Pi into operation you need:

- USB keyboard and mouse
- HDMI cable for monitor
- Network cable or WLAN
- MicroSD card with Raspbian operating system
- Micro-USB mobile phone charger as power supply unit (at least 1,500 mA)
- Audio cable for speakers (optional)

The power supply must be connected last, so that the Raspberry Pi switches on automatically. There is no separate on/off switch.

The Raspberry Pi 4

In June 2019, the Raspberry Pi 4 appeared with significantly more powerful hardware. For the first time, the Raspberry Pi with three different memory sizes: 1 GB, 2 GB and 4 GB. Further features in short are:

- Processor: Broadcom BCM2711, Quad core Cortex-A72 64-bit SoC 1.5 GHz
- Network: Gigabit Ethernet and Dual-Band IEEE 802.11ac WLAN
- USB ports: 2x USB 3.0, 2x USB 2.0 (distinguishable by colour)
- HDMI connections: 2x Micro HDMI. For the first time two monitors are supported.
Micro HDMI cables or adapters are required for connection.
- Power supply: USB type C port, 3A minimum required.

The slot for MicroSD cards, the 40-pin GPIO pin header and the jack socket for stereo audio and composite video are the same as the previous model.
Use the USB 2.0 ports for keyboard and mouse. The USB 3.0 ports are better used for USB sticks and external hard drives.

Operating system installation in a nutshell

For all those who do not yet have their Raspberry Pi ready for operation with the current Raspbian version, here is the system installation in ten steps:

1. Download NOOBS (at least version 3.0.0) of **www.raspberrypi.org/downloads** to your PC and unzip the zip archive to your hard disk.

2. If the SD card has already been used, reformat with the SD formatter in the PC: **www.sdcard.org/downloads/formatter_4**. Switch on **Format Size Adjustment** (the SD card must be at least 8 GB in size).

3. Copy all files and subdirectories from NOOBS to the SD card.

4. Remove the SD card from the PC, insert it into Raspberry Pi and boot. Select **English** as the installation language at the bottom. This automatically selects the English keyboard as well.

5. Check the preselected Raspbian operating system and click **Install** at the top left. After confirming a security prompt that the memory card is being completely overwritten, the installation starts, which takes a few minutes.

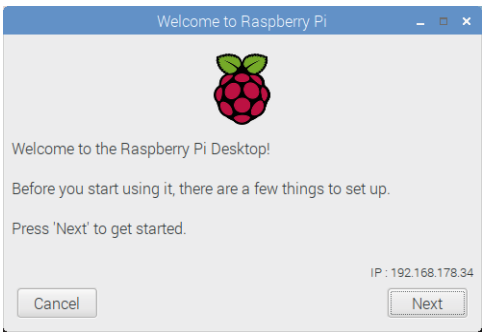
6. Once the installation is complete, the Raspberry Pi reboots.

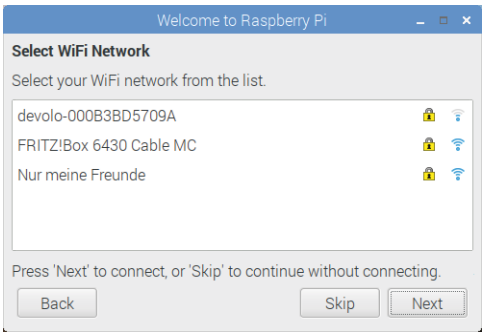
7. On the Raspbian desktop, the Configuration Wizard starts and displays the IP address of the Raspberry Pi. In the first dialog box, click **Next** and select Language and Time Zone if they are not automatically set to English.

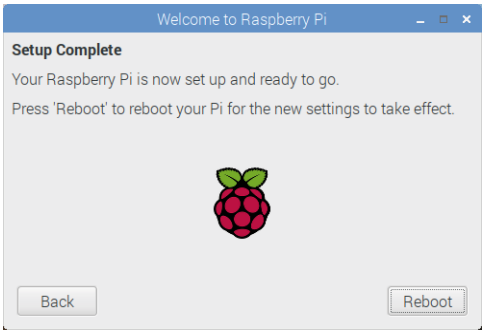
8. In the next step, it is recommended to change the default password, which is not absolutely necessary for the experiments in this learning pack. The default user **pi** is automatically logged in when booting, so you will rarely need the password.

9. For WLAN connection, select network and enter the password, for Ethernet connection simply click on **Skip**.

10. Finally, automatically download updates and restart the Raspberry Pi.







Num Lock key
As with almost all Linux systems, the numeric keypad is switched off by default when Raspbian is started. Press the Num-Lock key on the keyboard to activate it.

Programs to download
The programs used in the Advent calendar can be downloaded here: **www.buch.cd**. Enter the code 15045-5 for this product in the input field.

Open the website directly using the pre-installed browser on the Raspberry Pi and download the zip file into the home directory `/home/pi`.

Start the file manager on the Raspberry Pi. This automatically displays the home directory at start-up. Right-click on the downloaded zip file and select Unzip here from the context menu.

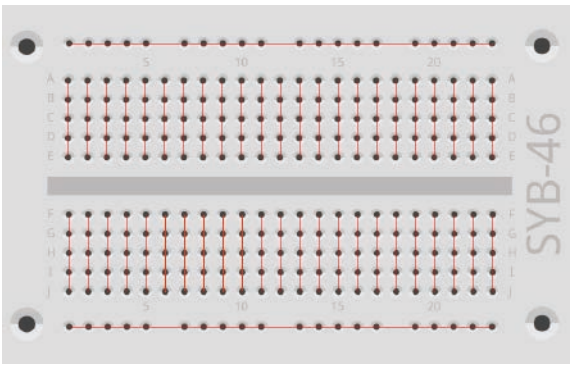
You will find this manual in the download archive of the Advent calendar as a colour PDF, so you can easily recognise the individual lines.

The most important components briefly explained

Pinboard

The Advent calendar contains a plug-in board for assembling electronic circuits quickly, without soldering. Here electronic components can be inserted directly into a hole grid.

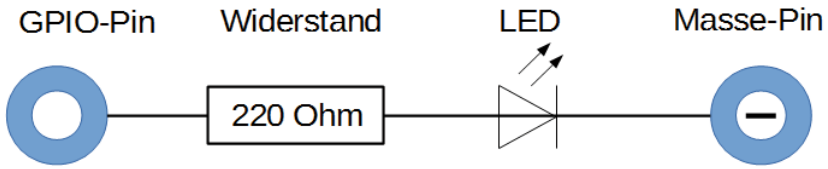
With this plug-in board, all outer longitudinal rows are connected to each other via contacts (X and Y). These contact rows are often used as positive and negative poles for the power supply of the circuits. In the other contact rows, five contacts (A to E and F to J) are connected crosswise, with a gap in the middle of the circuit board. This allows larger components to be inserted and wired to the outside.



The connections on the pinboard.

LEDs

LEDs (short for light emission diodes) light up when a current flows through them in the direction of flow. LEDs are represented in circuits with an arrow-shaped triangle symbol, which indicates the flow direction from the positive pole to the negative pole or to the ground line. An LED lets almost any current through in the direction of flow; it has a very low resistance. To limit the flow current and thus prevent the LED from burning out, a 220 ohm series resistor is usually installed between the GPIO pin used and the anode of the LED or between the cathode and ground pin. This resistor also protects the GPIO output of the Raspberry Pi from high currents.



Circuit diagram of an LED with series resistor.

Connect LED in which direction?
The two connecting wires of an LED have different lengths. The longer wire is the positive pole, the anode, the shorter one the cathode. Easy to remember: The plus sign has one stroke more than the minus sign, making the wire a little longer. In addition, most LEDs are flattened on the minus side, comparable to a minus sign. Also easy to remember: Cathode = short = flat edge.

Resistor

Resistors are used for limiting current on sensitive electronic components and as series resistors for LEDs. The unit of measurement for resistors is Ohm. 1,000 ohms correspond to one kilo-ohm, abbreviated kOhm. 1,000 kOhm corresponds to a megohm, abbreviated MOhm. The omega character × is often used for the unit ohm.

The coloured rings on the resistors indicate the resistance value. With a little practice, these are much easier to recognise than tiny numbers that can only be found on old resistors.

Most resistors have four such colour rings. The first two colour rings denote the digits, the third a multiplier and the fourth the tolerance. This tolerance ring is mostly gold or silver coloured - colours that do not appear on the first rings. Thus the reading direction is always unambiguous. The tolerance value itself hardly plays a role in digital electronics. Since the tolerance ring is almost always golden, it helps determining the reading direction for the rings. The table shows the meaning of the coloured rings on resistors. The 220 Ohm resistors for the LEDs have the colour code red-red-brown.

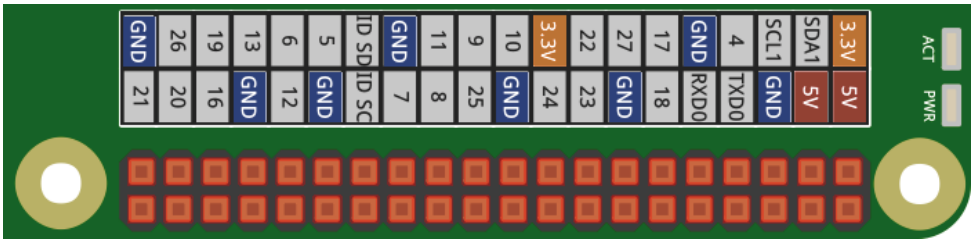
Colour	Resistance value in ohm			
	1. Ring (tens)	2. Ring (ones)	3. Ring (multiplier)	4. Ring (tolerance)
silver			10^{-2} = 0,01	±10 %
gold			10^{-1} = 0,1	±5 %
black		0	10^{0} = 1	
brown	1	1	10^{1} = 10	±1 %
red	2	2	10^{2} = 100	±2 %
orange	3	3	10^{3} = 1.000	
yellow	4	4	10^{4} = 10.000	
green	5	5	10^{5} = 100.000	±0,5 %
blue	6	6	10^{6} = 1.000.000	±0,25 %
violet	7	7	10^{7} = 10.000.000	±0,1 %
grey	8	8	10^{8} = 100.000.000	±0,05 %
white	9	9	10^{9} = 1.000.000.000	

It does not matter in which direction a resistor is installed. With LEDs, on the other hand, the installation direction is very important.

Even easier than using these tables you calculate the resistance value from a colour code using the freeware **Resistance Calculator** from **www.ab-tools.com/de/software/widerstandsrechner**. Conversely, you can also use it to display the colour code for any resistance value.

GPIO connection cable

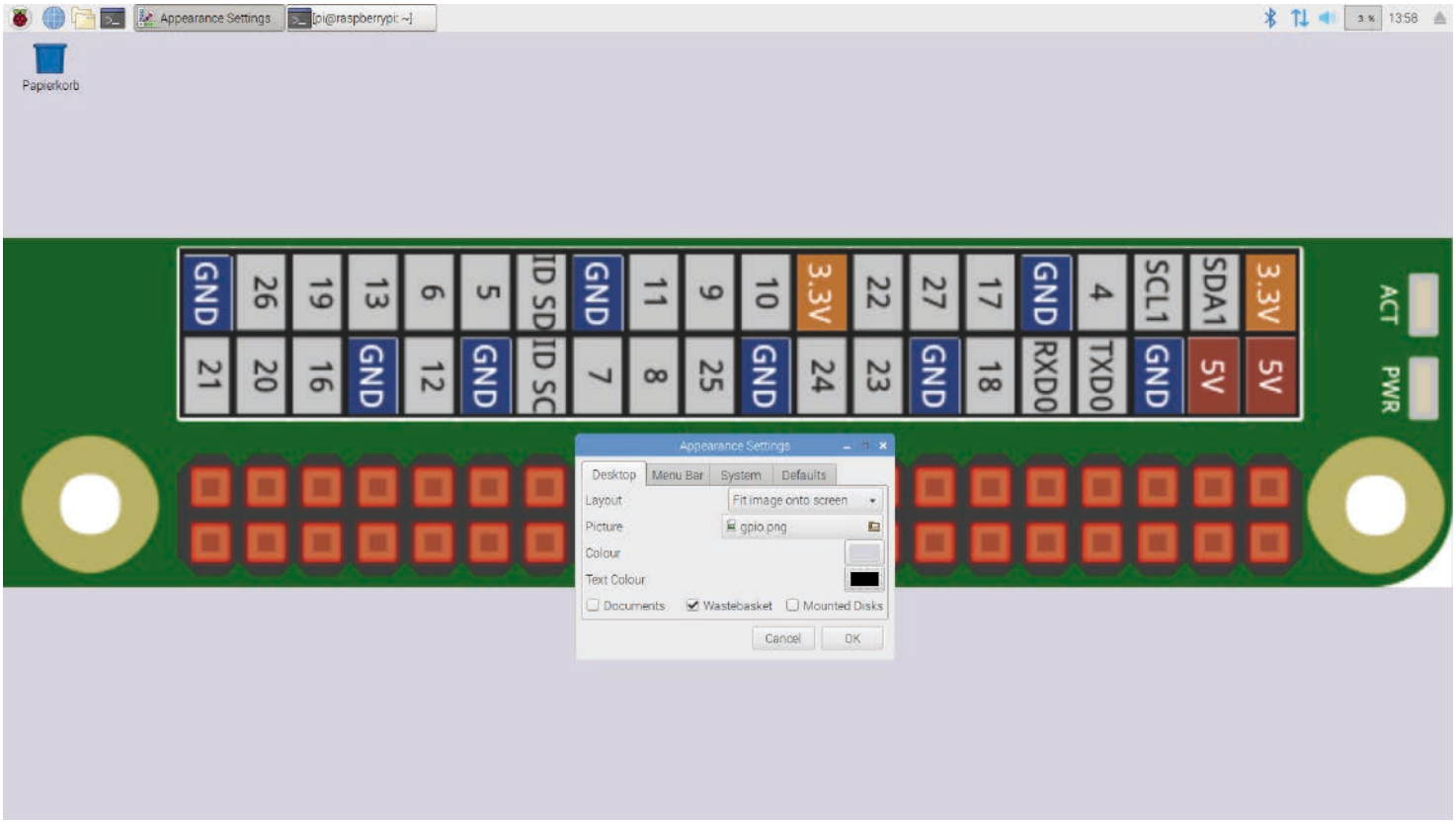
The coloured connection cables all have a plug on one side and a socket on the other side, which fits on a GPIO pin of the Raspberry Pi. The LEDs are also plugged directly into these sockets. The plugs are plugged into the plug-in board. The programmable GPIO pins on the Raspberry Pi have numbers, the ground pins are marked by GND in the figure.



Assignment of the GPIO pins.

This picture is included in the downloads for the Advent calendar. Save it on the Raspberry Pi to have access to it at any time, as only very few people can remember the pin numbers.

You can also simply set the image as your desktop background to keep it in view at all times. To do this, right-click on the desktop and in the next dialog box, click on the default image `road.jpg`. Select the file `gpio.png` from the folder `pi`. For **Layout** , select **Fit image onto screen**.



The Raspbian Desktop with the new background

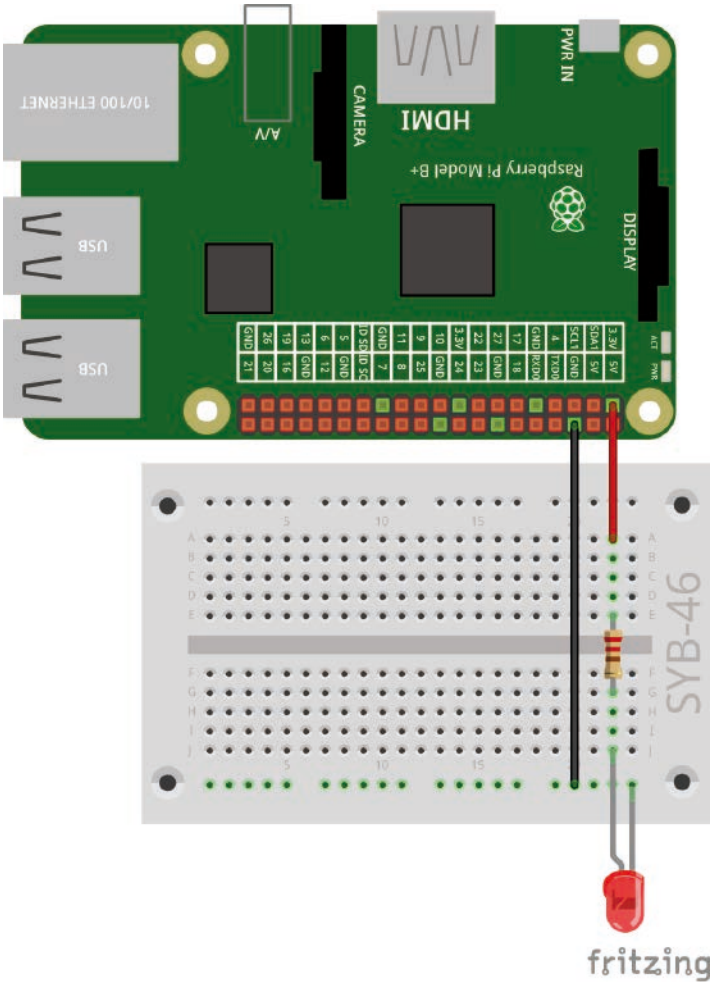
Precautions
You should under no circumstances connect any GPIO pins to each other and wait and see what happens.
Not all GPIO pins can be programmed freely. A few are fixed for power supply and other purposes. Some GPIO pins are directly connected to the processor terminals, a short circuit can completely destroy the Raspberry Pi. If two pins are connected to each other via a switch or an LED, a protective resistor must always be connected in between. LEDs with built-in series resistors are an exception.
For logic signals, always use pin 1, which supplies +3.3 V and can be loaded up to 50 mA. Pin 6 is the ground line for logic signals.
Pin 2 and 4 provide +5 V for power supply of external hardware. Here you can take as much power as the USB power supply of the Raspberry Pi supplies. However, these pins must not be connected to a GPIO input.

LED lights up

No program is needed for the first experiment. The Raspberry Pi serves here only as power supply for the LED. The experiment shows how LEDs are connected. Make sure that the LEDs are installed in the right direction. The flat side is in the picture to the right.

The most circuit configurations make use of the contact strip on one long side of the plug-in board as a ground contact. Inside, the cathodes of all LEDs are plugged in and connected by a cable to a GND pin on the Raspberry Pi.

Components: 1 plug-in board SYB-46, 1 LED red, 1 220-Ohm resistor (red-red-brown), 2 GPIO connection cables



The first LED lights up on the Raspberry Pi.

2rd day

Today in the Advent calendar

- 1 LED green
- 1 resistor 220 ohms
- 1 GPIO connection cable

Two LEDs flash alternately

The experiment of the second day causes two LEDs to light up alternating between red and green. The whole thing is controlled via an endless loop in Scratch.

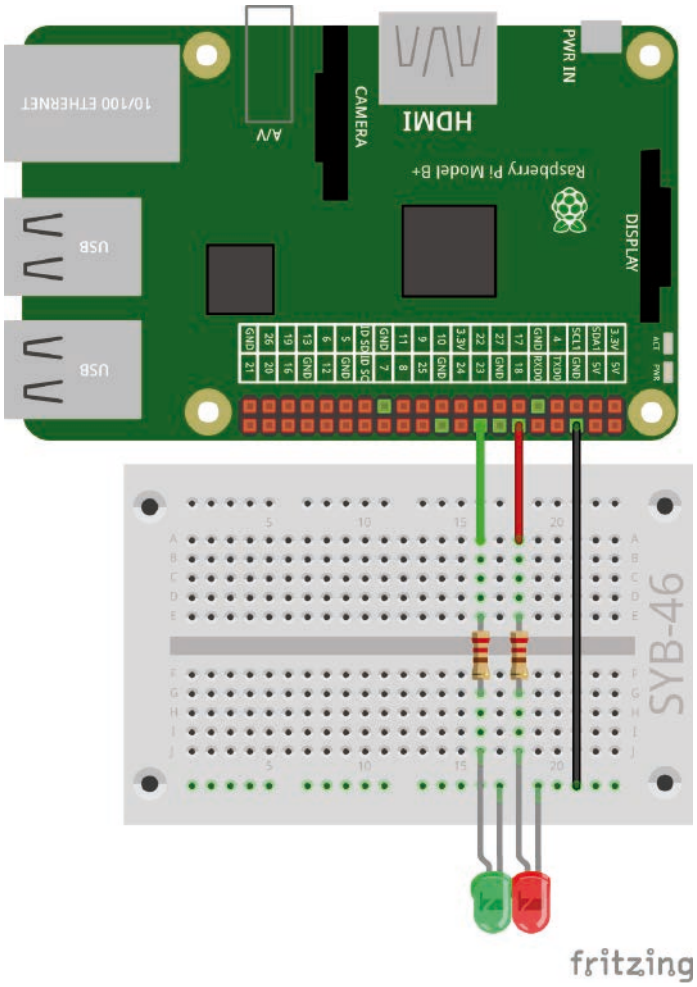
Components: 1 plug-in board SYB-46, 1 LED red, 1 LED green, 2 220-Ohm resistors (red-red-brown), 3 GPIO connection cables

This time the LED does not light up permanently, but is switched on by a program in the Scratch programming language for half a second.

Scratch is pre-installed on the Raspberry Pi in the menu - by clicking on the Raspberry logo in the top left corner - under **Development** and is considered one of the easiest to learn programming languages.

The new Scratch 2

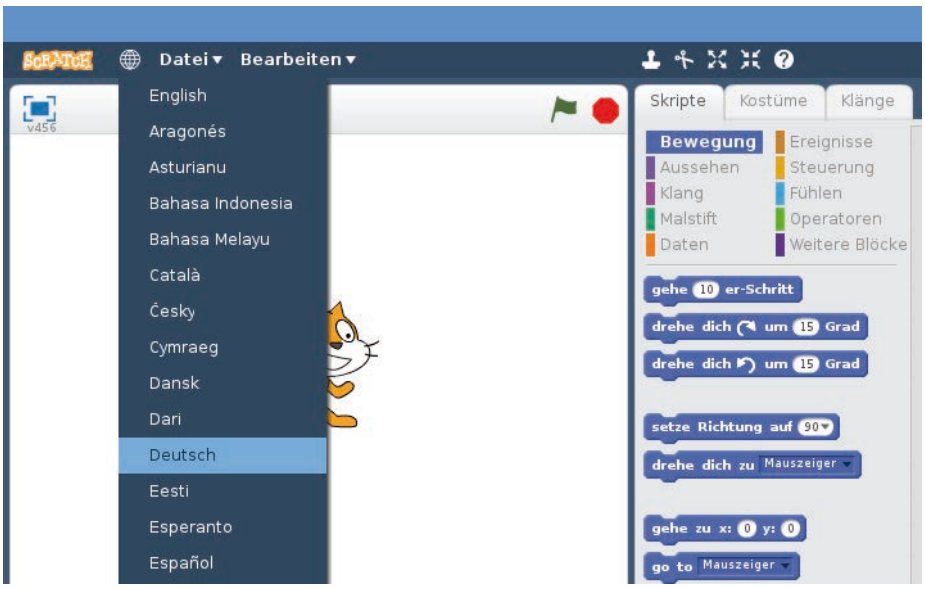
The Scratch programming language has been pre-installed in version 1.x ever since the first Raspbian version was released. The new version, Scratch 2, offers significantly more possibilities. Among other things, it allows you to create your own function blocks.
Scratch 2 runs online on the PC in the browser. However, this requires more computing power than a Raspberry Pi can currently offer. Since version NOOBS 2.4.0, a version of Scratch 2 is pre-installed in the Raspbian operating system, which runs offline without a browser and thus runs smoothly with the capabilities of a Raspberry Pi 3. With Scratch 2, hardware control via the GPIO interface has become much easier. For the Scratch projects in this Advent calendar we use the new Scratch 2 version from the menu **Development.**



Two LEDs flash on the Raspberry Pi.

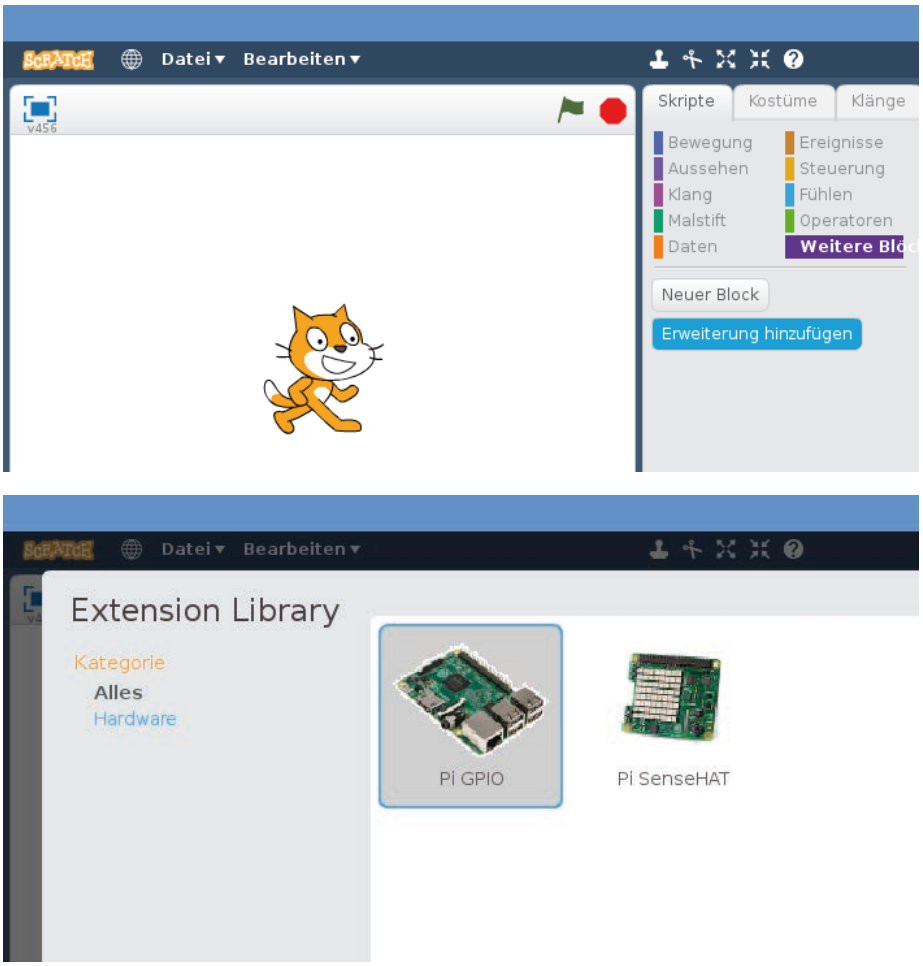
Basic settings for Scratch

Click on the globe at the top left of the Scratch logo and select **English**. The selected language remains saved, so you don't have to select it every time.



Switching Scratch 2 to English

Scratch 2 offers support for various hardware components on the GPIO port, which must be activated once via an add-on module. To do this, click **More blocks** on the block palette, and then click **Add extension**. Select the extension **Pi GPIO** here and install by double clicking.



Install Scratch 2 Pi-GPIO extension

The program

In Scratch you don't have to type any program code when programming. The blocks are simply attached to each other by dragging and dropping. The block palette in the middle of the Scratch window contains the available blocks sorted by type.

This Scratch program `02led02` flashes two LEDs alternately.

You can assemble the program on screen or use the program `02led02` from the download to the Advent calendar. To do this, from the **File/Load Project** menu, select the **pi** button in the next dialog box to select the personal home directory where the downloaded programs are located.

To assemble the program yourself, click on the yellow **Control** icon at the top of the block palette in the middle of the Scratch window. The blocks for Control are then shown in the block palette.

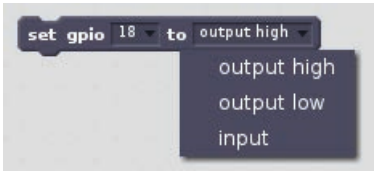
Drag the blocks you need from the block palette into the Scripts window on the right the Scratch screen.

The block **When (green flag) is clicked** is used to start a program. The block is round at the top, so it does not fit under any other block. It must always be set first.

The following script elements are executed when you click on the green flag in the upper right corner of the scratch background. The Scratch background is the part of the window on the top left with the cat. Interactive programs also play here.

A **forever** loop ensures that the two LEDs continue to flash endlessly until the user clicks on the red stop icon in the upper right corner of the Scratch background.

First, the red LED at pin 18 should be switched on and the green LED at pin 23 switched off. To do this, drag a block **set gpio ... to** ... from the PI-GPIO extension into the program and select pin **18** in the left list field. Select **output high** in the right list box to define this pin as output and switch it on.



In the next step, a further Scratch block **set gpio ... to** ... the green LED connected to GPIO pin 23 is switched off. To do this, select **output low** in the right list field of the block.

After the red LED on pin 18 is switched on and the green LED on pin 23 is switched off, the program waits half a second. For this, Scratch offers its own block **wait...secs**. Like many American programs, Scratch 2 uses the dot as decimal separator, not the comma used in other countries. So, half a second is entered as `0.5` and not as `0.5`.

The green LED at pin 18 is then switched on in the same way and the red LED at pin 8 is switched off. After another half a second, the cycle repeats from the beginning.

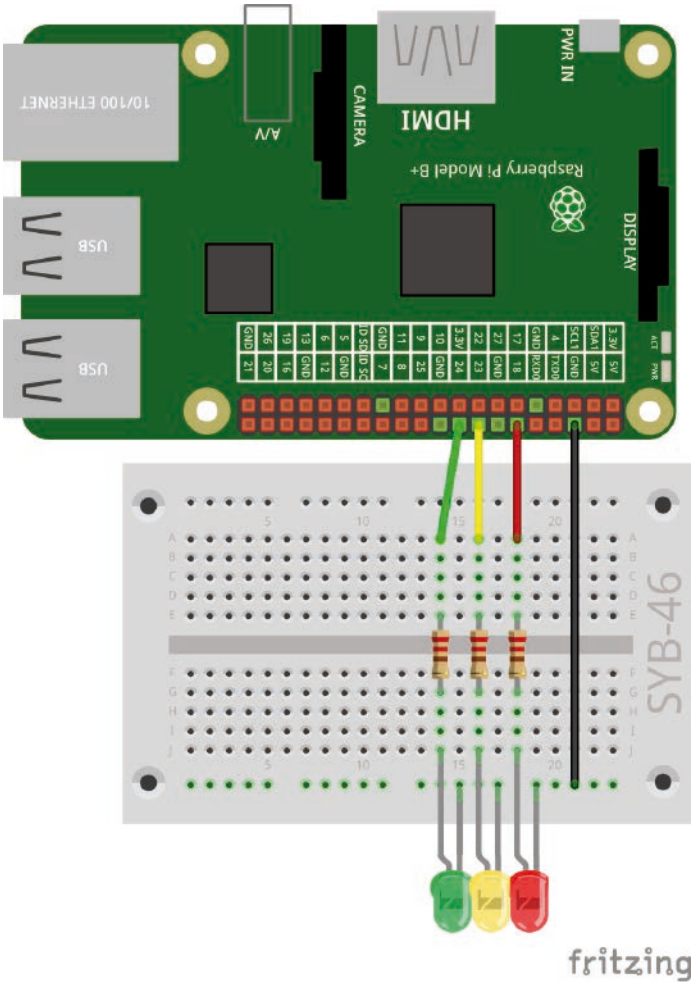
The program starts when you click on the green flag and ends when you click on the red stop symbol.

3. Day

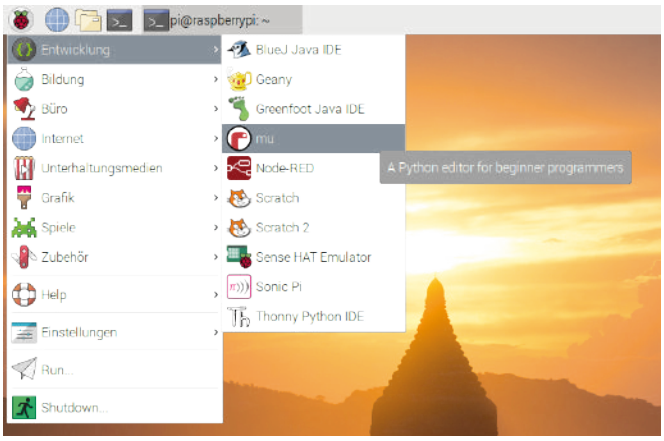
3rd day

Today in the Advent calendar

- 1 LED yellow
- 1 resistor 220 ohms
- 1 GPIO connection cable



A simple traffic light.



The cide editor Mu in the menu.

Traffic light circuit with Python

A traffic light with its typical light cycle from green to yellow to red and then back to green via a light combination red-yellow is easy to set up with three LEDs and shows how to control the GPIO pins on the Raspberry Pi in the popular programming language Python.

Components: 1 plug-in board SYB-46, 1 LED red, 1 LED yellow, 1 LED green, 3 220-Ohm resistors (red-red-brown), 4 GPIO connection cables

The programming language Python is pre-installed on the Raspberry Pi for an introduction to programming. Python is ideal because of its clear structure, which allows an easy introduction to programming, but is also an ideal language to automate "times fast" something that would otherwise be done by hand. Since there are no variable declarations, types, classes or complicated rules to consider, programming is really fun.

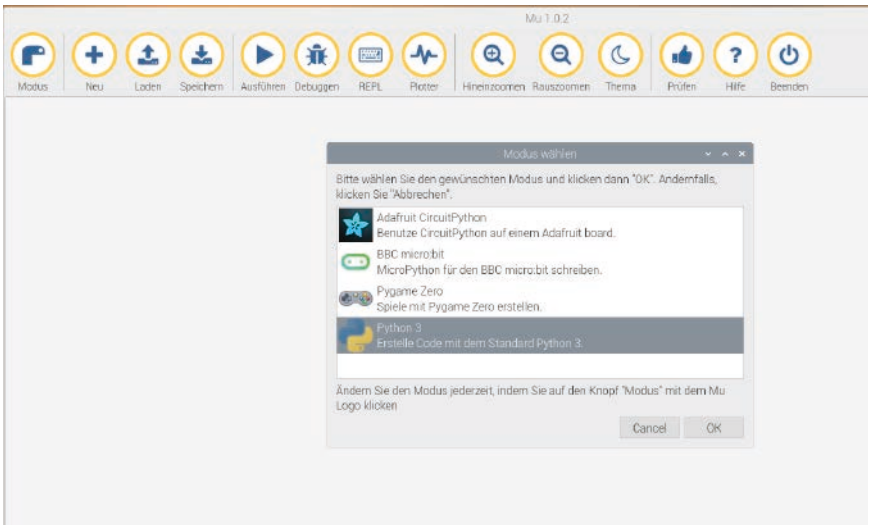
Python 3 with the code editor Mu

Since the operating system version NOOBS 3.1.1, which was specially developed for the Raspberry Pi 4, the new code editor **Mu** in the menu **Development** has been pre-installed instead of the classic Python development environment IDLE. Mu is a complete development environment. No additional components are required to start programming.

When Mu starts for the first time, select **Python 3**mode. Via the symbol **mode** you can change to another programming language at any time.

Python Flashcards

Python is the ideal programming language for learning the basics of programming. Only the syntax and the layout rules need some getting used to. The most important syntax elements of the Python language are briefly described in the form of small "cheat sheets" to help in everyday programming. These are based on the Python flashcards by David Whale. What this is all about can be found at **bit.ly/pythonflashcards3**. These flashcards do not explain the technical background, but only describe the syntax, i.e. how something is done, using very short examples.



The first launch of Mu.

BEDINGUNGEN 8

```
a=1
if a==1:
     print("gleich")
if a!=1:
     print("nicht gleich")
if a<1:
     print("kleiner")
if a>1:
     print("größer")
if a<=1:
     print("kleiner oder gleich")
if a>=1:
     print("größer oder gleich")
```

python 3 V1 (deutsch) – softwarehandbuch.de

IF ELSE 9

```
alter=10
if alter>17:
     print("Du darfst Auto fahren")
else:
     print("Du bist nicht alt genug")
```

python 3 V1 (deutsch) – softwarehandbuch.de

IF ELIF ELSE 10

```
alter=10
if alter<4:
     print("Du bist in der Kinderkrippe")
elif alter<6:
     print("Du bist im Kindergarten")
elif alter<10:
     print("Du bist in der Grundschule")
elif alter<19:
     print("Du bist im Gymnasium")
else:
     print("Du hast die Schule verlassen")
```

python 3 V1 (deutsch) – softwarehandbuch.de

AND/OR BEDINGUNGEN 11

```
a=1
b=2
if a>0 and b>0:
     print("Beide sind nicht Null")

if a>0 or b>0:
     print("Mindestens eine ist nicht Null")
```

python 3 V1 (deutsch) – softwarehandbuch.de

Extract from the Python flashcards

The program

Instead of slowing down with programming theory, algorithms and data types, we write the first small program in Python.

In Mu's main window, type in the program code shown. You can see interactive help texts as you type. Mu recognises the words and standard functions of the Python programming language and displays tips for use.

The line:

```
# Write your code here :-)
```

is, like all lines beginning with a #character, only a comment and can be deleted easily.

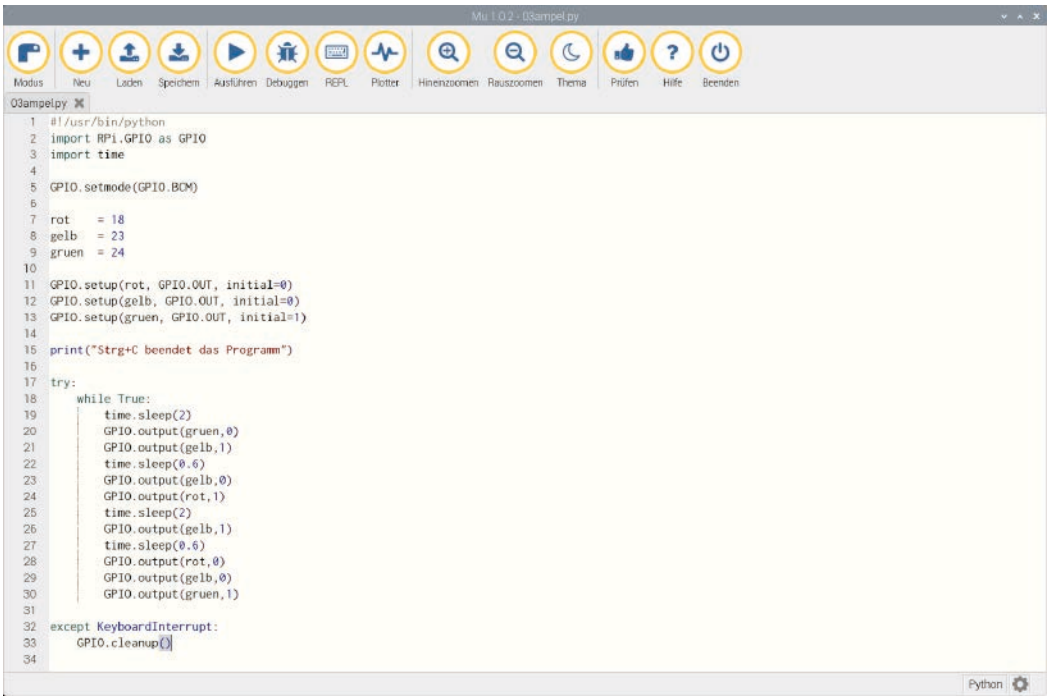
Save the file with the symbol **Save** as `03ampel.py`. Or you can use **File/Open** to open the finished program file from the download in the Python shell. The colour coding in the source code appears automatically and helps to find typos.

03ampel.py - /home/pi/03ampel.py (3.5.3)

File Edit Format Run Options Window Help

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

rot    = 18
gelb   = 23
gruen  = 24

GPIO.setup(rot, GPIO.OUT, initial=0)
GPIO.setup(gelb, GPIO.OUT, initial=0)
GPIO.setup(gruen, GPIO.OUT, initial=1)

print("Strg+C beendet das Programm")

try:
    while True:
        time.sleep(2)
        GPIO.output(gruen,0)
        GPIO.output(gelb,1)
        time.sleep(0.6)
        GPIO.output(gelb,0)
        GPIO.output(rot,1)
        time.sleep(2)
        GPIO.output(gelb,1)
        time.sleep(0.6)
        GPIO.output(rot,0)
        GPIO.output(gelb,0)
        GPIO.output(gruen,1)

except KeyboardInterrupt:
    GPIO.cleanup()
```

Das erste Programm in Python 3



The first program in Python 3

Start the program with the symbol **Execute**.

How it works

It is easy to test whether the program works. Now, of course, some questions will arise: What happens in the background? What do the individual program lines mean?

This first project shows the basic elements of Python and the `RPi.GPIO` library.

```
#!/usr/bin/python
```

Python programs that are started from the command line must always start with the top line. This is not necessary for programs that are only started via the Python shell. For compatibility reasons, however, you should get used to entering this line at the beginning of every Python program.

```
import RPi.GPIO as GPIO
```

The library `RPi.GPIO` must be imported into every Python program in which it is to be used. Using this notation, all functions of the library can be addressed via the prefix `GPIO`.

```
import time
```

The frequently used Python library `time` has nothing to do with GPIO programming. It contains functions for time and date calculation, and among other things also a function `time.sleep()`, with which waiting times can be easily realised in a program.

```
GPIO.setmode(GPIO.BCM)
```

At the start of each program, you must define how the GPIO ports are designated. Usually the standard numbering `BCM` is used.

Numbering of GPIO pins

The library `RPi.GPIO` supports two different methods for pin identification. In mode `BCM` the known GPIO port numbers are used, which are also used on command line level or in scratch. In the alternative, largely unused mode `BOARD`, the designations correspond to the pin numbers from 1 to 40 on the Raspberry Pi board, counted sequentially.

```
red = 18
yellow = 23
green = 24
```

These three lines define variables with the GPIO pins to which the three LEDs are connected. This makes the program clearer, and it is easier to adapt to a different circuit design, since the pin numbers only have to be changed at one point in the program.

```
GPIO.setup(red, GPIO.OUT, initial=0)
GPIO.setup(yellow, GPIO.OUT, initial=0)
GPIO.setup(green, GPIO.OUT, initial=1)
```

The three GPIO pins used are initialised one after the other as outputs. We do not use GPIO port numbers, but the previously defined variables.

The `GPIO.setup()` instruction may contain an optional parameter `initial` which assigns a logical state to the GPIO pin at initialisation. With this we switch on the green LED in this program right from the start. The other two LEDs start the program when it is switched off.

```
print("Ctrl+C ends the program")
```

Now a short operating manual is displayed on the screen. The program runs automatically. The key combination [Ctrl]+[C] should end it.

To query whether the user quits the program with [Ctrl]+[C], we use a `try...except` query. The program code entered under `try:` is initially executed normally. If during this time a system `except`ion occurs - it could be an error or the key combination [Ctrl]+[C] - it is aborted and the except instruction at the end of the program is executed.

```
except KeyboardInterrupt:
  GPIO.cleanup()
```

This key combination triggers a `KeyboardInterrupt` and automatically exits the loop. The last line closes the used GPIO ports and switches off all LEDs. The program is then terminated.

By closing the GPIO ports in a controlled manner, there are no system warnings or abort messages that could confuse the user.

The actual traffic light cycle runs in an endless loop:

```
    while True:
```

Such endless loops always require a termination condition, otherwise the program would never be terminated.

```
        time.sleep(2)
```

The green LED lights up for 2 seconds at the beginning of the program and also at each new start of the loop.

```
        GPIO.output(green,0)
        GPIO.output(yellow,1)
        time.sleep(0.6)
```

Now the green LED is switched off and the yellow LED is switched on. This then lights up alone for 0.6 seconds.

```
        GPIO.output(yellow,0)
        GPIO.output(red,1)
        time.sleep(2)
```

Now the yellow LED is switched off again and the red LED is switched on. This then lights up alone for 2 seconds. The red phase of a traffic light is usually much longer than the yellow phase.

```
        GPIO.output(yellow,1)
        time.sleep(0.6)
```

At the start of the red-yellow phase, the yellow LED is additionally switched on without another LED being switched off. This phase lasts 0.6 seconds.

```
        GPIO.output(red,0)
        GPIO.output(yellow,0)
        GPIO.output(green,1)
```

At the end of the loop the traffic light jumps back to green. The red and yellow LEDs are then switched off, and the green LED is switched on. The loop starts again in the green phase of the traffic light with a waiting time of 2 seconds. Of course you can adjust all times as you like. In reality, the traffic light phases depend on the dimensions of the intersection and the traffic flows. The yellow and red-yellow phases are usually 2 seconds long each.

This loop is repeated until the user presses [Ctrl]+[C].

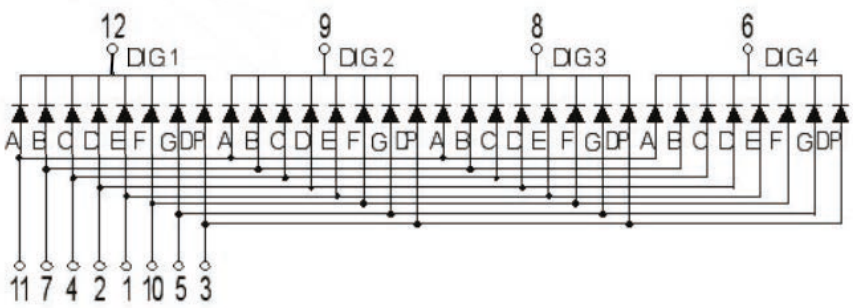
4. Day

4th day

Today in the Advent calendar

· Seven-segment display

Today, the Advent calendar contains a four-digit seven-segment display with which numbers and simple symbols can be displayed. Such seven-segment displays are often used in digital clocks, and previously also in pocket calculators. Petrol stations use this display technology in a large format. However, mechanical display elements are often used here instead of LEDs. Not all letters of the alphabet can be displayed clearly on these displays, but the letters A to F, which are necessary to display hexadecimal numbers, do work. Most seven-segment displays still have an eighth LED for the decimal point.

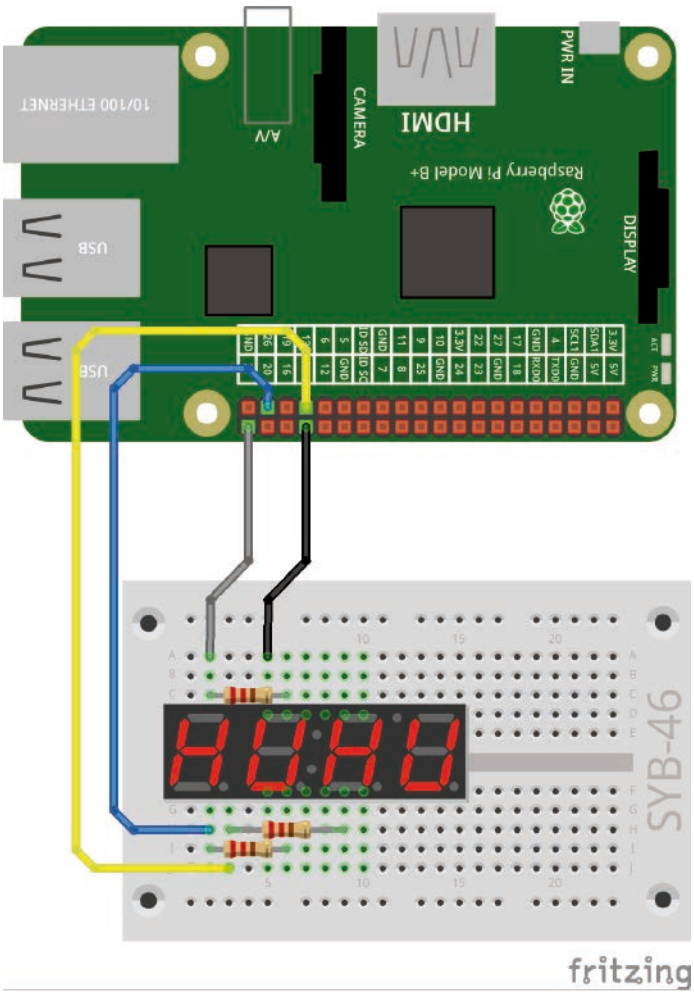


Circuit diagram of the seven-segment display.

With a single-digit seven-segment display, each LED can be controlled individually via its anode. All LEDs use a common cathode or vice versa. Accordingly, the display modules are referred to as "Common Cathode" or "Common Anode".

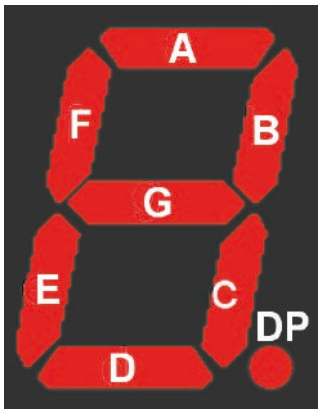
In the four-digit seven-segment display supplied with the Advent calendar, the cathodes of the eight LEDs of a digit are connected together according to the common cathode circuit diagram. In addition, the four anodes of the same LED position are connected together in all four digits. This means that the entire display for a total of 32 LEDs only needs 12 connections instead of 64.

The connections of the anodes are marked with the four digits 1 to 4, the connections of the cathodes after the seven segments with A to G as well as DP for the decimal point.



3 segments of the seven-segment display flash alternately.

The connections on the seven-segment display.



The names of the segments

To illuminate a single LED on the display, the cathode of the corresponding digit must be connected to the ground line and the anode of the segment must be connected to a +3.3 V signal. As usual, a 220 ohm series resistor must be connected in front of each LED.

Three segments of the seven-segment display will flash alternately

As a first simple example, the three horizontal segments of the seven-segment display will flash alternately.

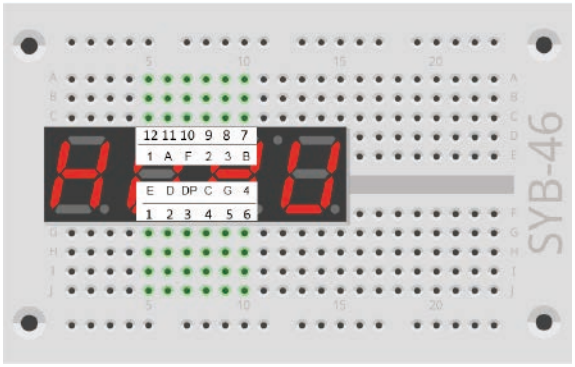
Components: 1 plug-in board SYB-46, 1 seven-segment display, 3 220-ohm resistor (red-red-brown), 4 GPIO connection cables

Wiring diagram of the seven-segment display

Once the seven-segment display is installed, the connections are difficult to see. It is installed on the plug-in board in rows 5 to 10 in all experiments in this Advent calendar.

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pins are not used in this program.

In this program only the three horizontal segments A, G and D are connected to GPIO pins via resistors. The digit 1 is connected to GND to let the segments of the first digit flash.



Seven-segment display with connection numbers on the plug-in board

The program

The program `04seg7.py` causes the three LEDs of the horizontal segments A, G and D to flash alternately one after the other.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg=[21, 13, 26]
for s in range(3):
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

print("Ctrl+C ends the program")
try:
  while True:
    for i in range(3):
      GPIO.output(seg[i], 1)
      time.sleep(0.2)
      GPIO.output(seg[i], 0)

except KeyboardInterrupt:
  GPIO.cleanup()
```

Pin seven-segment display	Pinboard	Segment / digit	**GPIO pin**
1	5 F-J	E	–
2	6 F-J	D	26
3	7 F-J	DP	–
4	8 F-J	C	–
5	9 F-J	G	13
6	10 F-J	4	–
7	10 A-E	B	–
8	9 A-E	3	–
9	8 A-E	2	–
10	7 A-E	F	–
11	6 A-E	A	21
12	5 A-E	1	GND

How it works

The first lines are already known, they import the libraries `RPi.GPIO` for controlling the GPIO ports and `time` for time delays. The numbering of the GPIO ports is then set to BCM as in the previous example.

```
seg=[21, 13, 26]
```

A list containing the GPIO numbers of the pins used is set up to control the three LEDs.

```
for s in range(3):
  GPIO.setup(seg[s], GPIO.OUT, initial=0)
```

The three GPIO ports used are initialised one after the other as outputs. We do not use GPIO port numbers, but the previously defined list. Within a list, the single elements are indexed by numbers starting with 0. `seg[0]` is therefore the first element, in this case `21`. The parameter `range()` in the `for` loop specifies how often the loop is run. More precisely, it denotes the first value that is no longer reached.

Each `for` loop requires a loop counter, a variable that takes a new value each time it passes through the loop. This value can be queried like any other variable within the loop. In the three loop passes, the variable `s` assumes the values 0, 1 and 2 and thus initialises the three elements of the list. For endless flashing, like in the last program, we use a `try: ... except`-construction with an endless loop, which is terminated by the key combination [Ctrl]+[C].

```
  while True:
    for i in range(3):
      GPIO.output(seg[i], 1)
      time.sleep(0.2)
      GPIO.output(seg[i], 0)
```

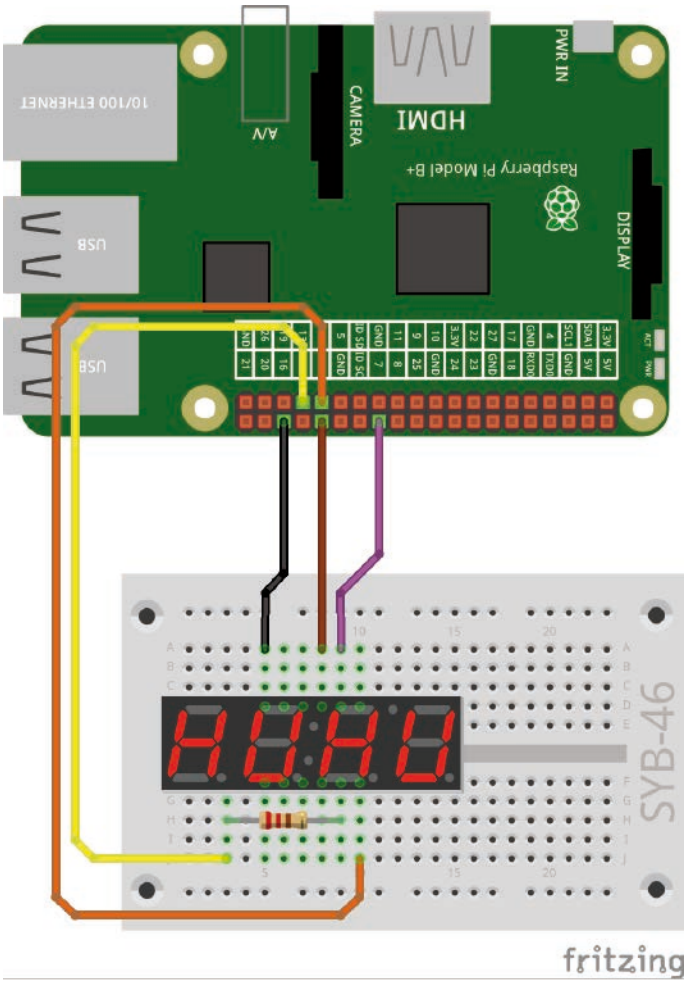
Within the infinite loop, another loop runs three times and switches on one of the segments in each pass, waits 0.2 seconds and switches it off again. The next segment in the list then follows.

5. Day

5th day

Today in the Advent calendar

- 1 GPIO connection cable



One segment and four digits of the seven-segment display connected.

Running light on the seven-segment display

The experiment of the 5th day causes the middle segment of all four digits of the seven-segment display to light up as a running light. For this purpose, the program switches the cathodes of the digits alternately, while the anode for the segment always remains switched on.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 1 220-ohm resistor (red-red-brown), 5 GPIO connection cables

This circuit uses only a single series resistor connected to the common cathode. It does not matter whether the series resistor is connected to the anode in front of the LED or to the cathode behind the LED. Make sure, however, that each LED has its own series resistor. In this example, only one LED lights up at a time, so there is no problem here.

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pins are not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F–J	E	–
2	6 F–J	D	–
3	7 F–J	DP	–
4	8 F–J	C	–
5	9 F–J	G	13
6	10 F–J	4	6
7	10 A–E	B	–
8	9 A–E	3	7
9	8 A–E	2	12
10	7 A–E	F	–
11	6 A–E	A	–
12	5 A–E	1	16

The program

The program `05seg7.py` switches on the G-segments of all four digits one after the other in an endless loop and switches them off again after 0.2 seconds each, resulting in a chaser effect.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg=13
digit=[16, 12, 7, 6]

GPIO.setup(seg, GPIO.OUT, initial=1)
for z in range(4):
  GPIO.setup(digit[z], GPIO.OUT, initial=1)

print("Ctrl+C terminates the program")
try:
  while True:
    for i in range(4):
      GPIO.output(digit[i], 0)
      time.sleep(0.2)
      GPIO.output(digit[i], 1)

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

The program works with some small differences like the program of day 4.

Segment G at pin 13 is permanently switched on. The cathodes of the four digits are defined in the list `num[]` and are briefly set to 0 one after the other in the infinite loop. In state 0 the cathode is connected to ground; the segment lights up. The cathodes of the other three segments are set to 1 for as long as they are connected to +3.3 V. The respective segments do not light up.

6th day

Today in the Advent calendar

- 3 GPIO connection cable

Another running light on the seven-segment display

The experiment of the 6th day causes all segments of the first digit of the seven-segment display to light up as a running light. This is possible with only one series resistor by means of a circuit trick.

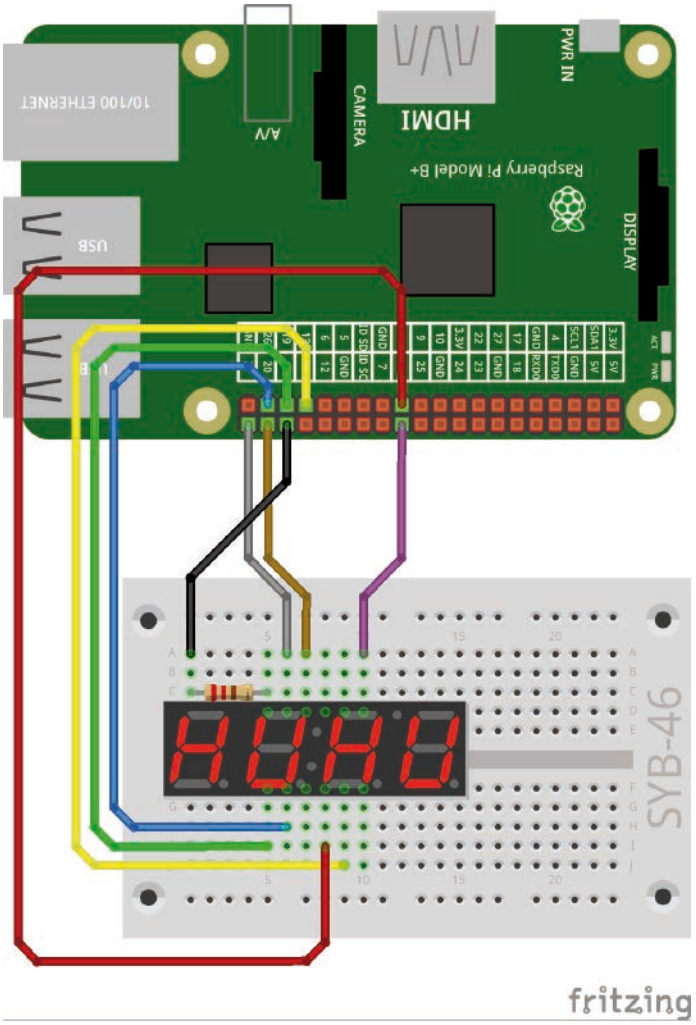
Components: 1 plug-in board SYB-46, 1 seven-segment display, 1 220-ohm resistor (red-red-brown), 8 GPIO connection cables

This circuit also uses only a single series resistor connected to the common cathode.

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pins are not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	–
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	–
7	10 A-E	B	8
8	9 A-E	3	–
9	8 A-E	2	–
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16



Seven segments and one digit connected to the seven-segment display.

The program

The program `06seg7.py` switches the segments of a digit on one after the other and off again after 0.1 seconds. The running light effect looks like an 8. The segment G in the middle is switched on twice in each pass.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=16
GPIO.setup(num, GPIO.OUT, initial=0)

print("Strg+C ends the program")
try:
  while True:
    for s in "abgedcgf":
      GPIO.output(seg[s], 1)
      time.sleep(0.1)
      GPIO.output(seg[s], 0)
```

```
except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

A variable of type `dictionary`, a special form of the list, is used for the seven segments of the display. In a `dictionary` , the individual elements are not selected by their number, but by any word, the so-called key, which can also consist of a single letter. In contrast to a simple list, a `dictionary` is enclosed in brackets. It can contain any number of pairs of key and value.

```
seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
```

In our case, the key is the respective code letter for the segment, the value behind it is the GPIO pin used.

```
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)
```

The GPIO pins of all seven segments are defined as outputs and set to `0`, which means switched off for an anode. The loop processes all letters of the specified string one after the other, instead of incrementing them within a certain number range, as is usually the case.

```
digit=16
GPIO.setup(num, GPIO.OUT, initial=0)
```

The cathode of the digit used is set to 0 to switch on the digit.

```
    for s in "abgedcgf":
      GPIO.output(seg[s], 1)
      time.sleep(0.1)
      GPIO.output(seg[s], 0)
```

Each time the endless loop is run, an inner loop processes the entered character string, sets the corresponding segments of the display to `1` one after the other and switches them on for 0.1 seconds.

If the user presses the key combination [Ctrl]+[C], the infinite loop is terminated as in previous experiments, and the GPIO ports are closed.

7th day

Today in the Advent calendar

- 1 resistor 220 ohms
- Jumper wire

Jumper wire

Today, jumper wire is included in the Advent calendar. The jumper wire is used to create short connection bridges, which are used to connect rows of contacts on the plug-in board. Depending on the experiment, cut the wire to the appropriate length using a small wire cutter. To be able to insert the wires easier into the plug-in board, it is advisable to cut the wires slightly diagonally to create a kind of wedge.

Four digits show the same flashing pattern

The program of the 7th day shows a blinking pattern with four segments on all four digits. The program shows you how to achieve a different effect with minimal changes based on an earlier - general - program.

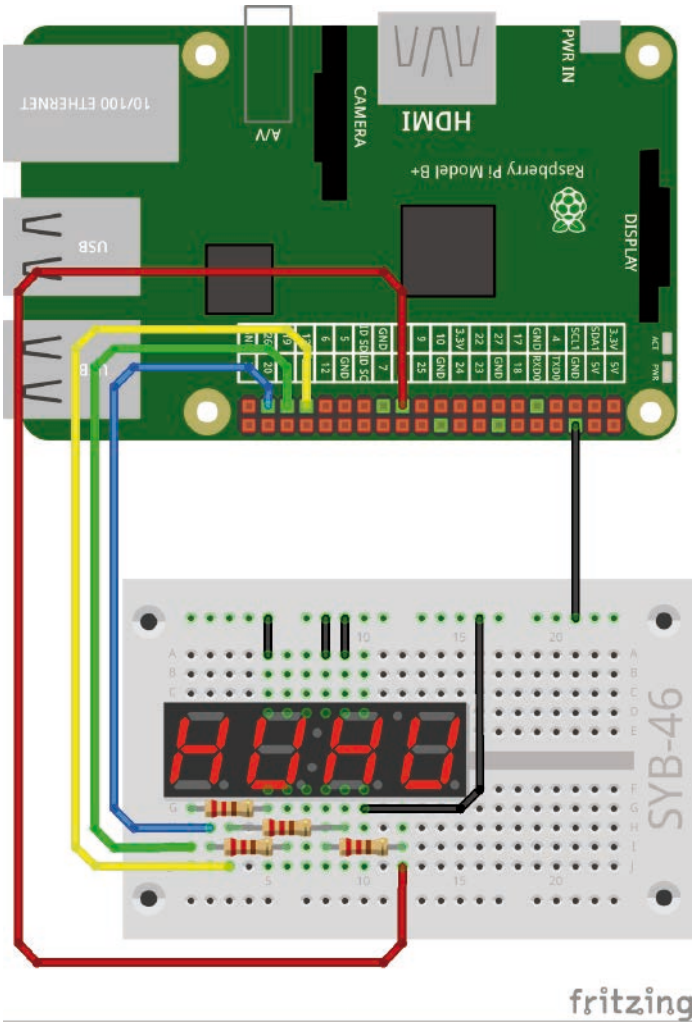
Components: 1 plug-in board SYB-46, 1 seven-segment display, 4 220-ohm resistor (red-red-brown), 5 GPIO connection cables, 4 wire jumpers (different lengths)

The four cathodes of the digits are connected via jumper wires to the upper contact strip of the plug-in board shown in the illustration. This is connected to a GND pin of the Raspberry Pi.

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pins are not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	GND
7	10 A-E	B	
8	9 A-E	3	GND
9	8 A-E	2	GND
10	7 A-E	F	
11	6 A-E	A	
12	5 A-E	1	GND



Four segments of the seven-segment display connected. The digits are connected to ground.

The program

The program `07seg7.py` runs a running light on all four digits over the lower four segments (C, D, E, G).

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=16
GPIO.setup(num, GPIO.OUT, initial=0)

print("Strg+C ends the program")
try:
  while True:
    for s in "cdeg":
      GPIO.output(seg[s], 1)
      time.sleep(0.1)
      GPIO.output(seg[s], 0)

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

The program differs from the previous one only in a single line:

```
    for s in "cdeg":
```

This line indicates which segments should light up in the loop. All definitions and the program logic remain the same. Owing to the parallel connection of the digits, no further GPIO pins need to be initialised.

For this special circuit, the general program can be simplified by omitting unneeded definitions, as the program `07seg7_02.py` shows.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'c':11, 'd':26, 'e':19, 'g':13}
for s in "cdeg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

print("Strg+C ends the program")
try:
  while True:
    for s in "cdeg":
      GPIO.output(seg[s], 1)
      time.sleep(0.1)
      GPIO.output(seg[s], 0)

except KeyboardInterrupt:
  GPIO.cleanup()
```

8th day

Today in the Advent calendar

· 3 resistors 220 ohms

Controlling a seven-segment display with Scratch

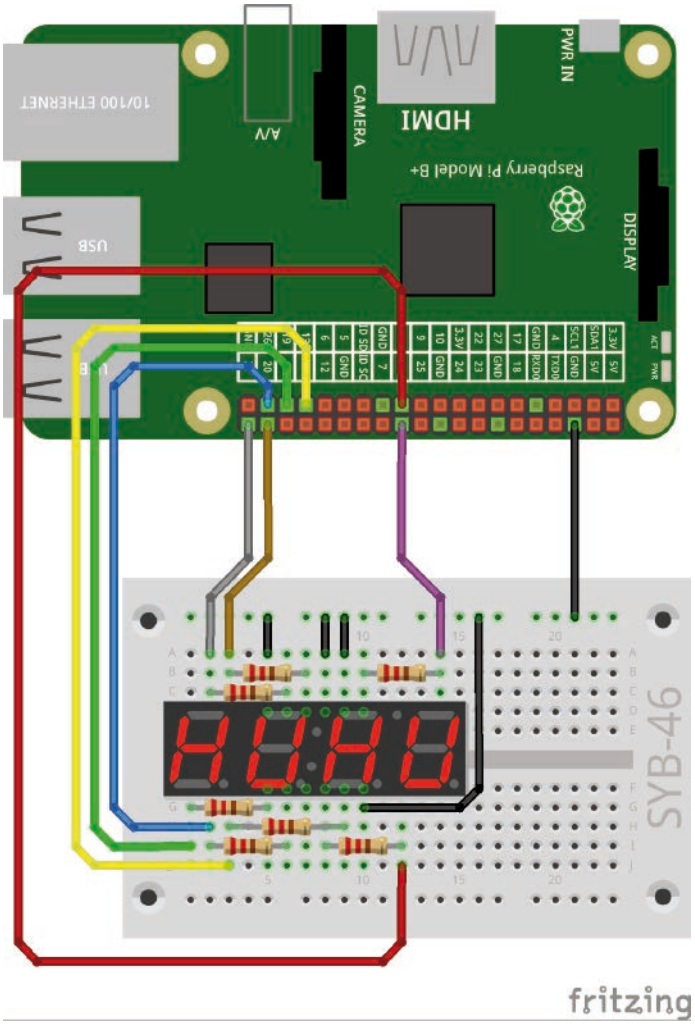
The program of the 8th day controls the segments of the seven-segment display interactively and individually with Scratch. New programming techniques from Scratch that have not yet been described are presented.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 7 220-ohm resistors (red-red-brown), 8 GPIO connection cables, 4 wire jumpers (different lengths)

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pin is not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	GND
7	10 A-E	B	8
8	9 A-E	3	GND
9	8 A-E	2	GND
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	GND



All seven segments of the seven-segment display connected. The digits are connected to ground.

The program

The Scratch-2 program `08seg7.sb2` offers a graphical interface to interactively switch the seven segments on or off individually.

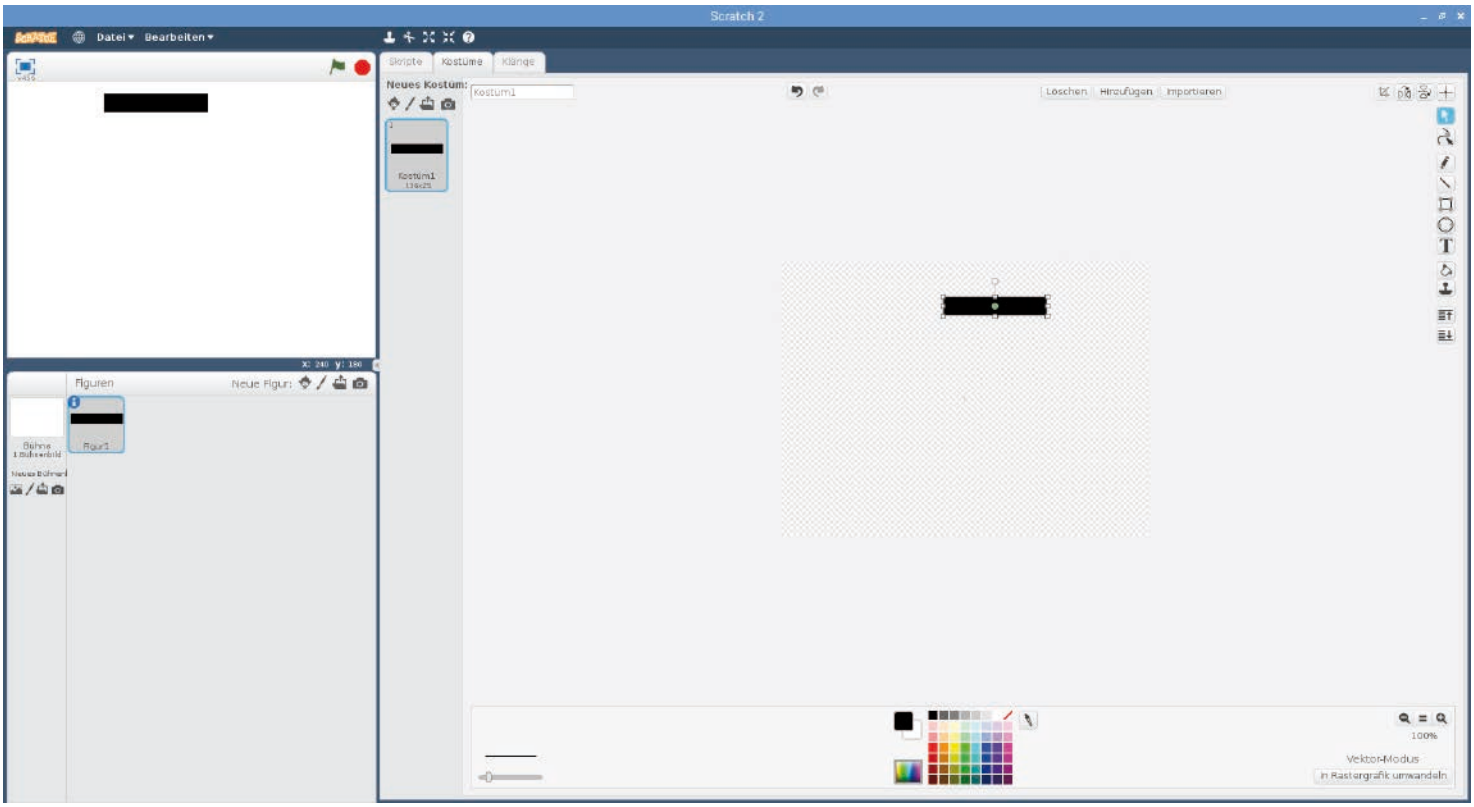
First delete the cat, the Scratch symbol figure, which is not used in this program. To do this, right-click on the cat in the lower right object/figure window and select **Delete** from the context menu.

The empty figure window in Scratch 2

In the program rectangular figures in the arrangement of the seven segments should switch the LEDs. You can paint this figure directly in Scratch. Click on the symbol **Draw new figure** in the figure window.

Scratch contains a simple paint program that you can use to paint figures. First finish one figure completely, you can duplicate the others later including all settings and script blocks and only adjust slightly.

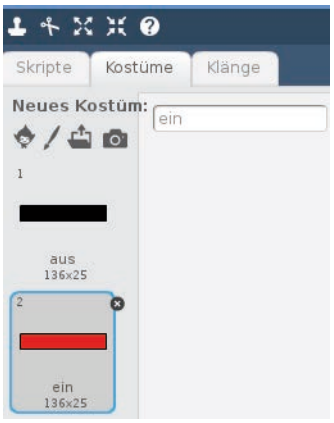
Switch the painting program to vector mode in the lower right corner. Then use the Rectangle tool in the toolbar on the right to draw a rectangle about the size shown. This is to represent the upper horizontal segment of the seven-segment display.



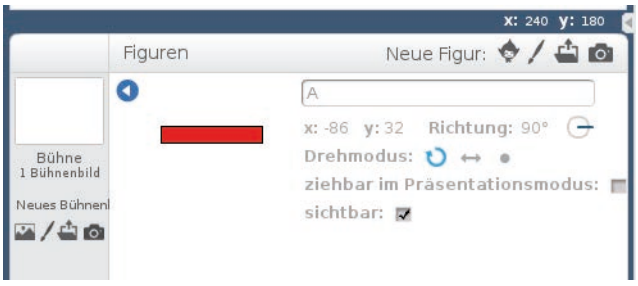
The first segment in Scratch 2's painting program

Each figure in Scratch can change its appearance using so-called costumes. Copy the existing costume. To do this, right-click it in the list of costumes and select **Duplicate** from the context menu. Rename the two costumes: **out** for the turned off LED and **on** for the turned on LED.

Edit costume **one** using the paint program. Fill the coloured area with the colour bucket icon with red to indicate the LED is on.



The two costumes of the first figure.



Costume for a switched off LED in the paint program.

Rename the figure in the figure palette at the bottom left to A because it represents the A segment of the seven-segment display. Click on the blue info symbol. A window then appears for renaming the current figure.

Scratch blocks for segment A of the seven-segment display

Next, create the program for this figure. Since the programs of all seven segments are very similar, you can duplicate them immediately using the figures and later only adjust them slightly.

Each figure gets its own script blocks. If the green flag is clicked, the GPIO pin 21 is set as output and switched off. To do this, attach the **When green flag** block **while clicking** a block **set gpio... to** ... .

If the figure itself is clicked, this segment will switch on, both on the screen and on the seven-segment display and switch off the next time it is clicked.

Use block **If LED1 is clicked** from the registration card **controller**.

In this case, the costume is first changed to the next one. This way the segment on the screen will switch with every click. Attach a block **next costume** from the block palette **Appearance**.

Then check a **if ... then ... otherwise** query, the costume of which is currently displayed. The costumes have the numbers 1 and 2. Depending on the active costume, the segment is switched on or off at GPIO pin 21.

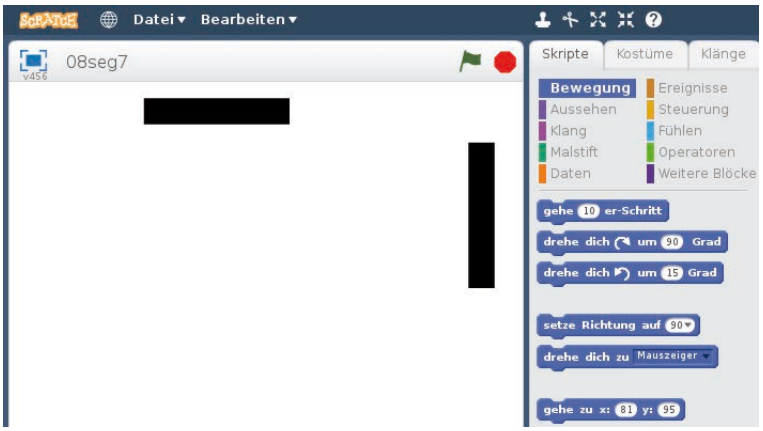
After all script blocks are finished, duplicate the figure **A** by right clicking the figure window. Rename the new figure to **B**.

Rotate the new figure by 90°, as it represents the vertical segment B. Don't turn the costumes, turn the whole figure. Switch to the **Scripts** tab and enter 90 degrees on the **Movement** block palette in the **Rotate clockwise ... on the block pallet**. You do not need to drag the block into the program. Simply double click on it, it will be executed, and the figure rotated. Then slide the figure to the appropriate position next to segment A.

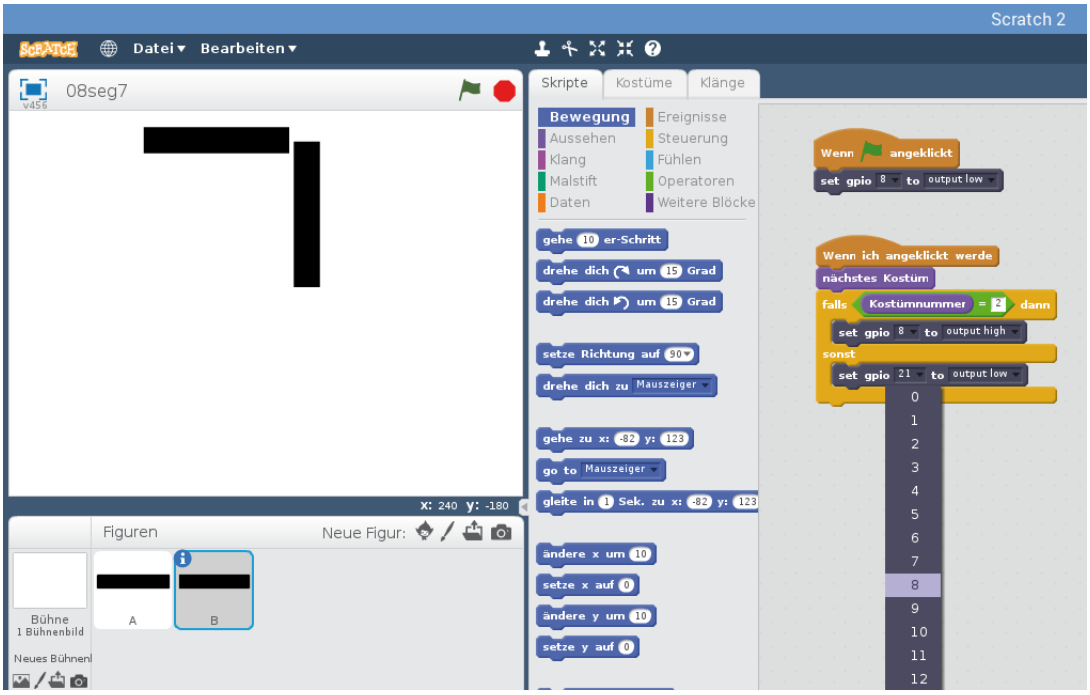
In all three **set gpio... to ...**-blocks select the GPIO pin 8 to which segment B of the seven-segment display is connected.

Duplicate the other figures in the same way, rename them, move them to the appropriate positions and change the GPIO pins.

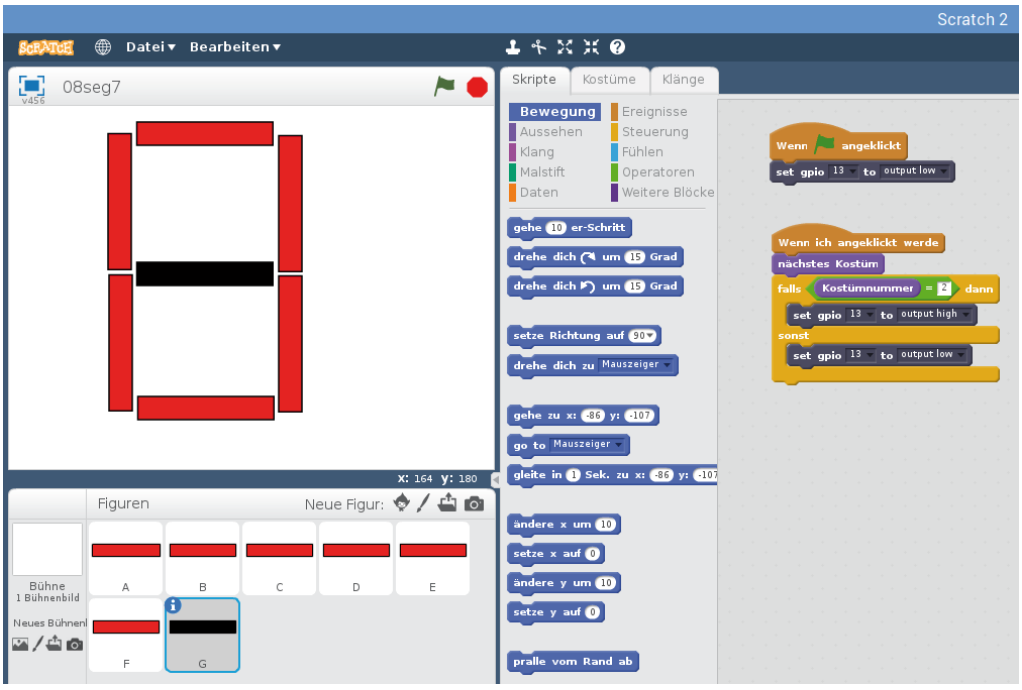
Start the program by clicking on the green flag. If you click on one of the figures, it will change colour and the corresponding LED of the seven-segment display is switched on or off.



Duplicate and rotate segment



Selecting GPIO Pins for Segment B



The finished program in action

9. Day

9th day

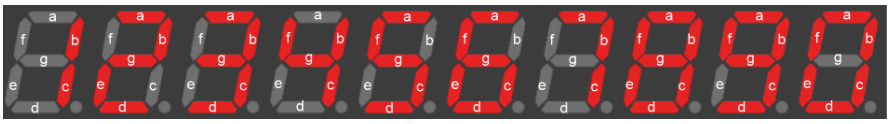
Today in the Advent calendar

- 1 GPIO connection cable

Digits on the seven-segment display

The actual purpose of a seven-segment display in most cases is not graphic gimmickry, but to display numbers. The program of the 9th day shows how this works.

The circuit design is the same as on the 8th day, but this time we use Python instead of Scratch.



The ten digit images of a seven-segment display.

The program

The program `09zahl.py` displays a counter, which counts all digits one after the other on the display. All four digits of the display always show the same number.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

print("Strg+C ends the program")
try:
  while True:
    for i in range(10):
      for s in number[i]:
        GPIO.output(seg[s], 1)
      time.sleep(0.5)
      for s in "abcdefg":
        GPIO.output(seg[s], 0)

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

In this program, the GPIO pins for the seven segments are also initialised using a dictionary.

```
number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3


  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]
```

The list `number` defines the segments to be switched on for the individual digits from 0 to 9. This is a normal list, of which the elements are only written below each other for the sake of clarity. The digits themselves are not part of the list. Like everything behind a `#` in a line, they are considered a comment by Python and are ignored during program execution.

The program runs again in an endless loop, which can be terminated by the user at any time with the key combination [Ctrl]+[C].

```
    for i in range(10):
```

A loop counts continuously from 0 to 9 to display the digits one after the other.

```
      for s in number[i]:
        GPIO.output(seg[s], 1)
```

Each time the endless loop passes, an inner loop processes string entered in the list for that `number`, sets the corresponding segments of the display to `1` and switches them on for 0.1 seconds.

```
      time.sleep(0.5)
      for s in "abcdefg":
        GPIO.output(seg[s], 0)
```

After the loop for a digit has completed a loop, it remains displayed for 0.5 seconds, then all seven segments are set to `0` regardless of their current status and are thus switched off. Then the display of the next number starts.

```
except KeyboardInterrupt:
  GPIO.cleanup()
```

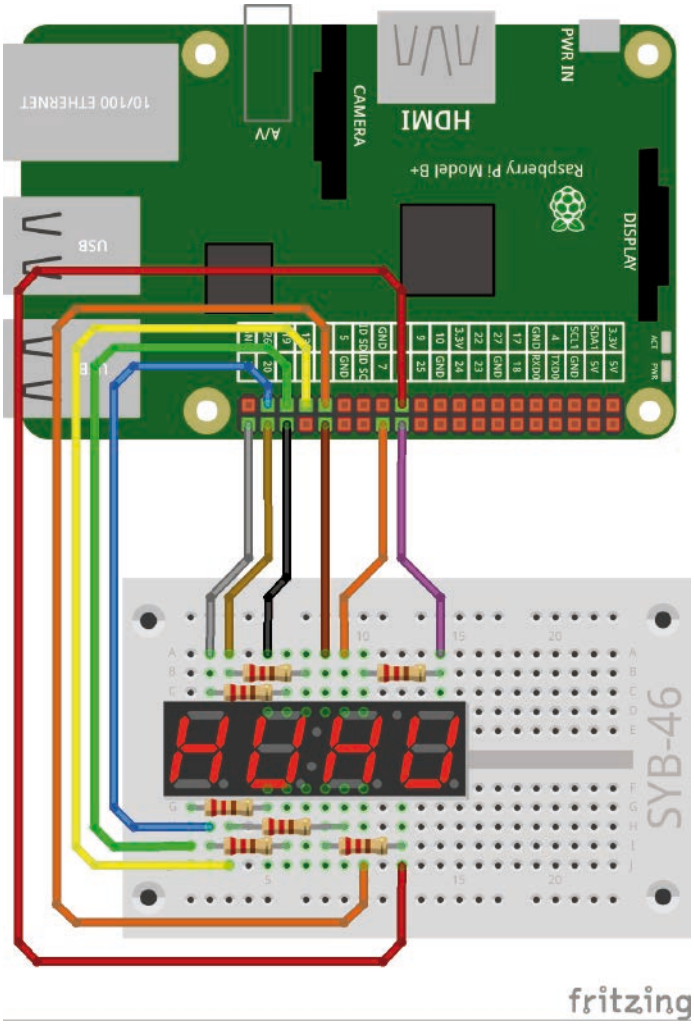
If the user presses the key combination [Ctrl]+[C], the infinite loop is terminated as in previous experiments, and the GPIO ports are closed.

10. Day

10th day

Today in the Advent calendar

· 1 GPIO connection cable



All seven segments of the seven-segment display connected. The digits are individually connected to GPIO pins.

Multiple digits on the seven-segment display

The previous experiment used all four digits of the seven-segment display simultaneously. It is much more interesting to use all four digits independently to represent different digits.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 7 220-ohm resistors (red-red-brown), 11 GPIO connection cables

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pin is not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

The trick with afterglow

Since each segment always has the same signal for all digits, individual digits can be switched on and off, but they cannot display different numbers. In order to display a different number on each number of the display, you have to use a trick. A light source is perceived by the human eye as still glowing for a short time, although it is actually already switched off.

The so-called time-division multiplexing procedure switches very quickly from one digit to the next and simultaneously displays a different number for each change. The trick is to find the right time interval. If the numbers change too quickly, they seem to blur when looking at it, if they change too slowly, a clear flickering can be seen.

Many such experiments found on the Internet also use an ATmega microcontroller, such as the Arduino, or a shift register module to control the seven-segment display. There are now also seven-segment displays in which such electronics are already installed. The program of the 10th day shows that control by time division multiplex is also possible without additional electronics besides the Raspberry Pi.

The program

The program `10zahl.py` displays the sequence of numbers `1234` on the seven-segment display.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)
```

```
digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

print("Strg+C ends the program")
try:
  while True:
    for i in range(4):
      for s in "abcdefg":
        GPIO.output(seg[s], 0)
      GPIO.output(digit[i], 0)
      for s in number[i+1]:
        GPIO.output(seg[s], 1)
      time.sleep(0.001)
      GPIO.output(digit[i], 1)

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

The initialisation of the GPIO ports, the list variables for the LEDs and the representation of the numbers were taken from the last experiment. The main loop that displays the numbers is new.

```
    for i in range(4):
```

Within the infinite loop, another loop runs four times to quickly display the four digits one after the other.

```
      for s in "abcdefg":
        GPIO.output(seg[s], 0)
```

A loop sets the anodes of all seven segments to `0`, thus switching them off. At this moment all LEDs are off.

```
      GPIO.output(digit[i], 0)
```

The cathode of the current digit is set to `0` so that this digit can then display the corresponding number.

```
      for s in number[i+1]:
        GPIO.output(seg[s], 1)
```

The next loop switches all segments of the number to be displayed to `1` and therefore on. This number is one higher than the loop counter, which counts from 0 to 3. The digits 1 to 4 will be displayed.

```
      time.sleep(0.001)
```

This line lets the program wait 1 millisecond. The LEDs of one digit in reality light up for exactly the same duration before switching to the next digit. Experiment with different waiting times to adjust the time division multiplex so that the display flickers as little as possible but does not blur.

```
      GPIO.output(digit[i], 1)
```

At the end of the loop, the cathode of the current digit is set to `1` and thus switched off. In the next loop pass, the next digit is displayed.

11. Day

11th day

Today in the Advent calendar

- 1 GPIO connection cable

The circuit design is the same as on the 10th day.

Show arbitrary numbers

In the last program, the number `1234` was fixed in the program. The next program offers the possibility to enter and display arbitrary numbers. The options for correcting user input errors are also shown.

The program

The program `11zahl.py` requires a four-digit number, but this does not mean that the user actually enters it. Therefore, the program must intercept all erroneous inputs or convert them to valid numbers. For example, if the user accidentally or intentionally enters a longer or shorter number or even letters and special characters, the program must not terminate because of the errors.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

z = [0,0,0,0]
print("Ctrl+C ends the program")

def za():
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)

try:
  while True:
    s = input("Please enter four-digit number:")
    s = s.zfill(4)
    for i in range(4):
      if s[i].isdigit():
        z[i] = int(s[i])
```

```
      else:
        z[i] = 0
    sek = time.time()
    while time.time() <= sec + 2:
      za()

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

Large parts of the program are known from the 10th day. Everything concerning the initialisation of the GPIO ports and the list for displaying the digits on the seven segments of the display is taken from the last program.

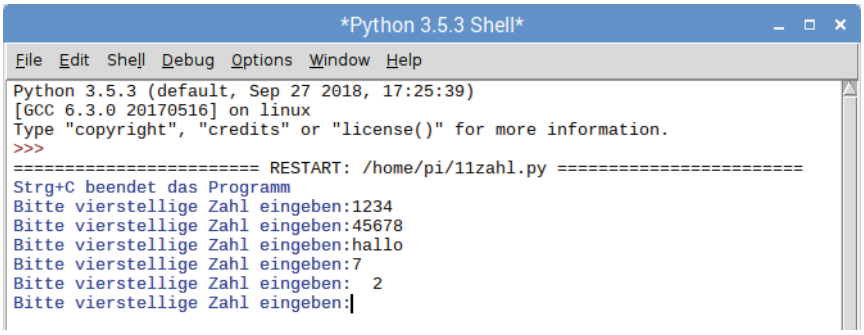
```
z = [0,0,0,0]
```

The list z[] contains four digits to be displayed in the course of the program. This list is initialised once at the beginning and filled with four zeros.

```
def za():
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)
```

The number display is now stored in a function `za()` , which is as general as possible and can therefore be used for other programs. It no longer displays the digits 1 to 4 one after the other, as in the last program, but the four digits from the list `z[]`.

The main part of the program is completely new and runs again as an endless loop. The user is prompted to enter a four-digit number. Whatever the user actually enters is converted into a valid four-digit number, which is then displayed for about 2 seconds. Then the prompt for entering a number appears again.



The program must not simply abort even if the input is invalid.

```
try:
  while True:
    s = input("Please enter four-digit number:")
```

The infinite loop is not used in this program to display the digits, but for user input. It therefore does not run many times per second, but only once for each number entered.

The user is prompted to enter a four-digit number. The function `input()` accepts the input in plain text without evaluating it and saves it as a string in the variable `s`. This character string can be of any length and can contain any other character besides digits. Invalid or missing numbers should simply be displayed as `0`.

```
s = s.zfill(4)
```

To mitigate the problem that the user may enter fewer than four characters or even simply press the [Enter] key, the string is filled in with zeros down to four digits at the beginning. The character string `s` is therefore always at least four characters long.

```
for i in range(4):
```

Now a loop begins which determines a valid digit for each of the four digits of the display and writes it to the list `z[]`. The first four characters of the character string `s` are evaluated. Starting with the fifth character, all characters are ignored.

```
  if s[i].isdigit():
    z[i] = int(s[i])
```

If the character read from the string is actually a digit, its numerical value is stored at the corresponding position in the list `z[]`. To check this, use the `isdigit()` method, which is automatically available for each string variable. It returns `True` if all characters of the string - which in this case consists of only one character - are digits.

The function `int()` converts a character or a floating point number into an integer. In this case, the character just read from the string is converted into the corresponding digit.

```
  else:
    z[i] = 0
```

In the other case, if there is another character at the position of the character string just read because the user did not adhere to the rules during input, a `0` is entered at the corresponding position in the list `z[]`.

The number defined in this way, the digits of which are now individually in four elements of the list `z[]` , shall be displayed for 2 seconds. Then the loop starts anew and prompts the user for input.

Since to display the number on the four-digit seven-segment display a separate program function must run permanently in quick succession, you cannot simply use `time.sleep()` for the waiting time. A `while` loop repeatedly runs the function `za()` for 2 seconds to display numbers and then ends the main loop.

```
sek = time.time()
while time.time() <= sec + 2:
  za()
```

In the variable `sek` the current time is stored in seconds, which the function `time.time()` outputs at any time to the nearest hundredth of a second.

When calculating time data in programs, it is best to always refer to the system time of the Raspberry Pi and not, for example, try to calculate the duration of a loop yourself and determine time periods from it. The background processes of a multitasking operating system can quickly falsify these results. The difference between two absolute times, on the other hand, is unambiguous.

The `while` loop queries whether two or more seconds have passed since the time was saved. As long as this is not the case, the function `za()` is called again and again to display numbers.

After the 2 seconds, this loop ends, and with it also the superior `while` loop ends. The display disappears, and the user is prompted to enter a new number.

12th day

Today in the Advent calendar

- 1 GPIO connection cable

The circuit design is the same as on the last days.

Countdown to Christmas

The program for the 12th day indicates how many days there are left until Christmas on the seven-segment display.

The program

The program `12countdown.py` uses some new Python elements for date calculation.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, datetime

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20,
'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

z=[0,0,0,0]
t=datetime.date(2019, 12, 24) - datetime.date.today()
z[0]=int(t.days/1000)
z[1]=int(t.days%1000/100)
z[0]=int(t.days/1000)
z[0]=int(t.days/1000)
print(z)
print("Ctrl+C terminates the program")

try:
  while True:
    for i in range(4):
      for s in "abcdefg":
        GPIO.output(seg[s], 0)
      GPIO.output(digit[i], 0)
      for s in number[z[i]]:
        GPIO.output(seg[s], 1)
      time.sleep(0.001)
      GPIO.output(digit[i], 1)

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

The initialisation of the GPIO pins and the list for displaying the digits on the seven segments of the display is taken from the last program.

```
import time, datetime
```

At the beginning, the library `datetime` for date calculations is imported additionally.

```
t=datetime.date(2019, 12, 24) - datetime.date.today()
```

The function `datetime.date()` creates a date object with a specific date. This line calculates the difference between two date objects, the period between 24.12.2019 and today. The object with the result is stored in the variable `t`.

```
z[0]=int(t.days/1000)
z[1]=int(t.days%1000/100)
z[0]=int(t.days/1000)
z[0]=int(t.days/1000)
```

Then the four digits to be displayed are stored in the four elements of the list `z[]`. `t.days` returns the days in the date object. The function `int()` stores only the integer part of the calculation results.

- The first digit `z[0]` results from dividing the days by 1000.
- The second digit `z[1]` is calculated using the modulo operator`%`. It gives the indivisible remainder for an integer division. The result is divided by 100 and gives the second digit of the number to be displayed.
- The third digit `z[2]` is calculated in the same way.
- The fourth digit `z[3]` is the indivisible remainder of the division by 10.

```
print(z)
```

This line displays the four digits in the Python shell. It is not necessary for the calculation and can also be omitted. The number is then displayed on the seven-segment display using the known endless loop.

13. Day

13th day

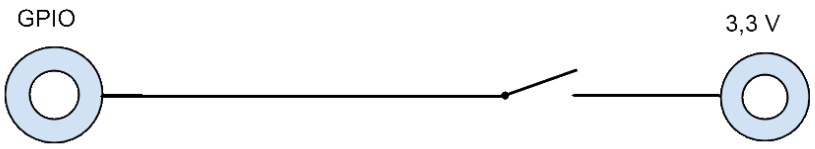
Today in the Advent calendar

- 1 button
- 1 resistor 10 kOhm

The connection diagram of the seven-segment display is the same as for the last days.

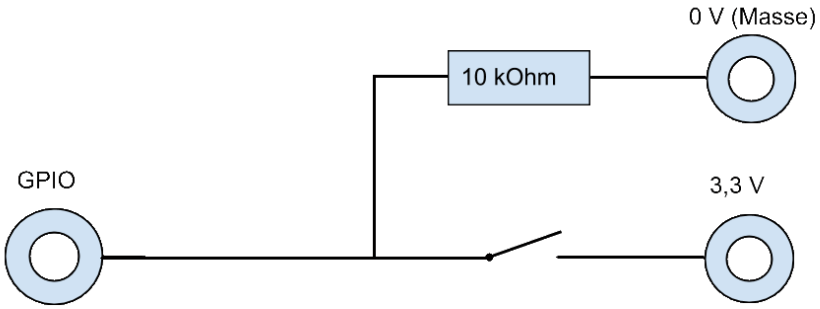
Connecting a push-button to a GPIO pin

Not only can GPIO pins output data, for example via LEDs, they can also be used for data input. For this they must be defined as inputs in the program. In the next project, we will use a button for input, which is plugged directly into the plug-in board. The pushbutton has four connection pins, whereby two opposite (large distance) pins are connected to each other. As long as the key is pressed, all four connections are connected. In contrast to a switch, a push-button does not engage. The connection is immediately disconnected again once released.



Push-button at a GPIO input

If a +3.3 V signal is present on a GPIO pin defined as input, it is evaluated as logical `True` or `1`. This way you can connect the respective GPIO pin to the +3.3 V connector of the Raspberry Pi via a push button, which you must not do with older Raspberry Pi models! This would overload the GPIO pin. The Raspberry Pi 3 and Raspberry Pi 3 B+ have built-in protective resistors and therefore no externally connected protective resistors are required.



Push-button with pull-down resistor at a GPIO input.

In most cases, this simple circuit already works, but the GPIO pin would not have a clearly defined state when the button is open. If a program queries this port, random results may occur. To prevent this, a comparatively very high resistance - usually 10 kOhm - is connected to ground. This so-called pull-down resistor pulls the status of the GPIO pin down to 0 V again when the push-button is open. As the resistance is very high, there is no danger of a short circuit as long as the button is pressed. When the button is pressed, +3.3 V and the ground line are connected directly via this resistor.

Christmas countdown with button

The program for the 13th day indicates how many days there are left until Christmas. Pressing the button starts a countdown which counts down to 0.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 7 220-ohm resistors (red-red-brown), 1 10 kOhm resistor (brown-black-orange), 1 push button, 14 GPIO connection cables

The contact strip on the left in the figure is connected to the GPIO pin 18 and via a 10 kOhm pull-down resistor (brown-black-orange) to the ground line on the lower contact rail of the plug-in board. The contact strip on the right in the illustration is connected to the +3.3 V line of the Raspberry Pi.

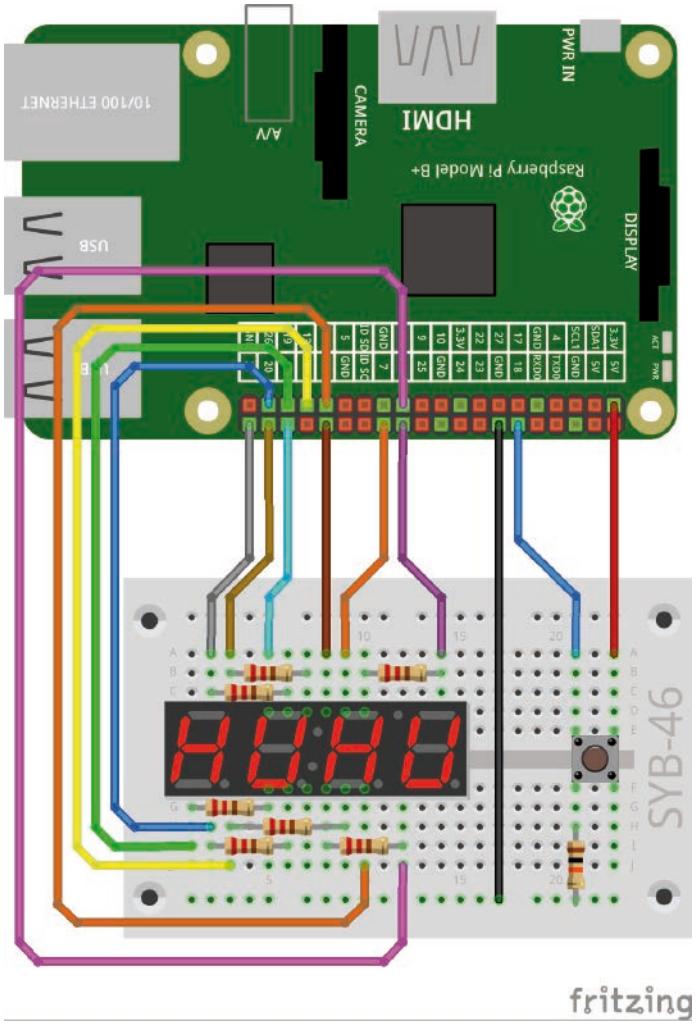
The circuit design of the seven-segment display is the same as for the previous days.

The program

The program `13countdown.py` again shows the number of days until Christmas. When the button is pressed, a countdown will run down to 0.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, datetime

GPIO.setmode(GPIO.BCM)
```

```
seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

button=18
GPIO.setup(button, GPIO.IN, GPIO.PUD_DOWN)

z=[0,0,0,0]

def za():
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)

t=datetime.date(2019, 12, 24) - datetime.date.today()
x=t.days
z[0]=int(x/1000)
z[1]=int(x%1000/100)
z[2]=int(x%100/10)
z[3]=int(x%10)

print("Press button to start countdown")
while GPIO.input(button)==0:
  za()

while x>=0:
  z[0]=int(x/1000)
  z[1]=int(x%1000/100)
  z[2]=int(x%100/10)
  z[3]=int(x%10)
  for j in range(100):
    za()
  x-=1

GPIO.cleanup()
```



Push-button with pull-down resistor.

How it works

The button at pin 18 is defined as GPIO input. The pin number is stored in the variable `button`.

```
button=18
GPIO.setup(button, GPIO.IN, GPIO.PUD_DOWN)
```

The program again uses the function `za()` known from an earlier program to display a four-digit number on the seven-segment display.

```
t=datetime.date(2019, 12, 24) - datetime.date.today()
x=t.days
z[0]=int(x/1000)
z[1]=int(x%1000/100)
z[2]=int(x%100/10)
z[3]=int(x%10)
```

The number of days until Christmas is first stored in a new variable `x` and then as single digits in the four fields of the list `z[]`.

```
while GPIO.input(button)==0:
  za()
```

An endless loop calls the function `za()` again and again to display this number as long as the button is not pressed.

```
while x>=0:
  z[0]=int(x/1000)
  z[1]=int(x%1000/100)
  z[2]=int(x%100/10)
  z[3]=int(x%10)
  for j in range(100):
    za()
  x-=1
```

If the button has been pressed, a new loop will start, which counts down the value `x` by 1 in each pass. The values for the list `z[]` are recalculated each time. Then, an inner loop calls the function `za()` a hundred times in a row, causing each number to glow for a tenth of a second before the next number is displayed.

If 0 is reached by counting down, it still appears on the display. The GPIO ports are then closed and the program ends.

14th day

Today in the Advent calendar

- 1 button

The connection diagram of the seven-segment display is the same as for the last days.

Connecting push-buttons without external pull-down resistors

Current Raspberry Pi models have built-in pull-down resistors that can be switched on and off on the software side. This saves the external resistor. The button can be connected directly between +3.3V and the GPIO pin, as explained in the experiment of the 14th day.

Christmas countdown with two buttons

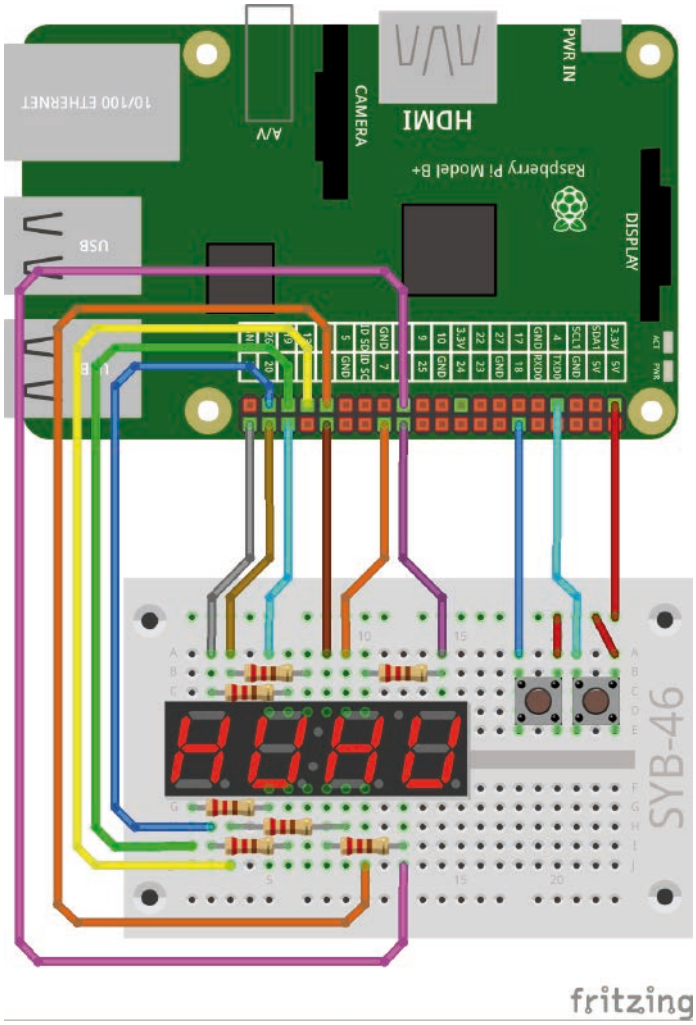
The program for the 14th day also indicates how many days there are left until Christmas. Pressing the button starts a countdown which counts down to 0. At the end, the countdown can be restarted using the same button. The second button ends the program. It must be pressed a little longer because, depending on where the program is at the moment, several loops have to be aborted one after the other.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 7 220-ohm resistors (red-red-brown), 2 buttons, 14 GPIO connection cables, 2 jumpers

The program

The program `14countdown.py` is based on the program of the 13th day and contains more loops and `break` instructions to abort the loops.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, datetime

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

button1=18
GPIO.setup(button1, GPIO.IN, GPIO.PUD_DOWN)
button2=4
```



Two push-buttons without external pull-down resistors

```
GPIO.setup(button2, GPIO.IN, GPIO.PUD_DOWN)

z=[0,0,0,0]

def za():
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)

print("Press button 1 to start countdown and restart")
print("Press 2 to quit")
while True:
  t=datetime.date(2019, 12, 24) - datetime.date.today()
  x=t.days
  z[0]=int(x/1000)
  z[1]=int(x%1000/100)
  z[2]=int(x%100/10)
  z[3]=int(x%10)
  if GPIO.input(taste2)==1:
    break

  while GPIO.input(taste1)==0:
    za()
    if GPIO.input(taste2)==1:
      break

  while x>=0:
    z[0]=int(x/1000)
    z[1]=int(x%1000/100)
    z[2]=int(x%100/10)
    z[3]=int(x%10)
    for j in range(100):
      za()
    x-=1
    if GPIO.input(taste2)==1:
      break

  while GPIO.input(taste1)==0:
    za()
    if GPIO.input(taste2)==1:
      break

GPIO.cleanup()
```

How it works

The initialisation of the GPIO pins and the list for displaying the digits on the seven segments of the display is taken from the last program.

```
button1=18
GPIO.setup(button1, GPIO.IN, GPIO.PUD_DOWN)
button2=4
GPIO.setup(button2, GPIO.IN, GPIO.PUD_DOWN)
```

The GPIO pins of the two buttons are then initialised. The parameter `GPIO.PUD_DOWN` at the initialisation of the GPIO pins for the push buttons turns on the built-in pull-down resistor. This eliminates the need for an external pull-down resistor.

The whole program now runs in an endless loop, because the countdown can be repeated again and again by pressing one of the
buttons.

```
while True:
  t=datetime.date(2019, 12, 24) - datetime.date.today()
  x=t.days
  z[0]=int(x/1000)
  z[1]=int(x%1000/100)
  z[2]=int(x%100/10)
  z[3]=int(x%10)
  if GPIO.input(taste2)==1:
    break
```

The infinite loop and also all subordinate loops check in each pass whether key 2 is pressed. If this is the case, the loop is aborted with a `break` statement.

```
  while GPIO.input(taste1)==0:
    za()
    if GPIO.input(taste2)==1:
      break
```

After calculating the number of days until Christmas, this number is displayed on the seven-segment display as long as button 1 is not pressed. This loop also queries button 2. If this is pressed, the loop is aborted. But then the next loop will follow. To cancel all loops up to and including the outer endless loop, press the button for a few milliseconds, similar to a reset button.

```
  while x>=0:
    z[0]=int(x/1000)
    z[1]=int(x%1000/100)
    z[2]=int(x%100/10)
    z[3]=int(x%10)
    for j in range(100):
      za()
    x-=1
    if GPIO.input(taste2)==1:
      break
```

If button 1 is pressed, the loop will end regularly, and the next loop will follow, counting down the countdown to 0. It can now also be aborted by pressing button 2.

```
  while GPIO.input(taste1)==0:
    za()
    if GPIO.input(taste2)==1:
      break
```

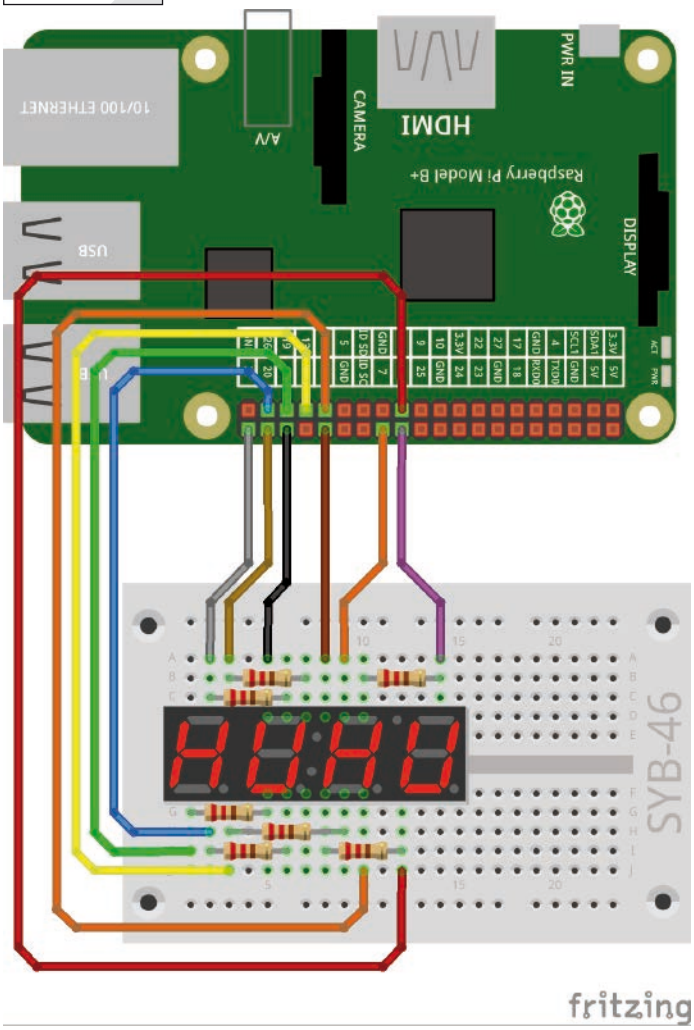
Finally, all four digits are set to 0. Another loop will display this number until button 1 is pressed. In this case, the endless loop starts again from the beginning and calculates the number of days remaining until Christmas.

15. Day

15th day

Today in the Advent calendar

- 1 GPIO connection cable



All seven segments of the seven-segment display connected. The digits are individually connected to GPIO pins.

The connection diagram of the seven-segment display is the same as for the last days.

Digits with Scratch on the seven-segment display

The program of the 15th day shows how Scratch numbers can be displayed on a seven-segment display. Scratch is unfortunately not fast enough to control all four digits of the display by time division multiplex, even in turbo mode. The program therefore only displays a single digit at a time.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 7 220-ohm resistors (red-red-brown), 11 GPIO connection cables

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pin is not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

Initialising GPIO pins for the digits

The infinite loop expects a user input at the beginning.

The program

The Scratch 2 program `15number.sb2` expects a digit from the user, which is then displayed on the seven-segment display.

The program uses the same circuit structure as the programs of the previous days. To avoid having to wire the cathodes of the individual segments to ground or +3.3V depending on their use, the program initially switches off the first three digits at GPIO pins 16, 12 and 7 to **High** and thus off and the right digit at GPIO pin 6 to **Low** and thus on.

The anodes of the seven segments do not need to be initialised in advance. They are automatically initialised when a digit is displayed.

Next, an infinite loop will start, which first expects input from the user.

The block **asks ... and wait** from the block palette **Feel** displays an input field on the Scratch background. The current figure, in the example, the cat, shows the question. The program does not continue until the user has entered something.

Ten blocks now ask for this input one after the other. The block **characters are used to ... from ...** from the block palette **Operators** in combination with the block ... = ... only considers the first character, since only one character can be displayed at a time.

The block **Answer** from the block palette **Feel** automatically contains the entered character string.

If the user has entered a 0...

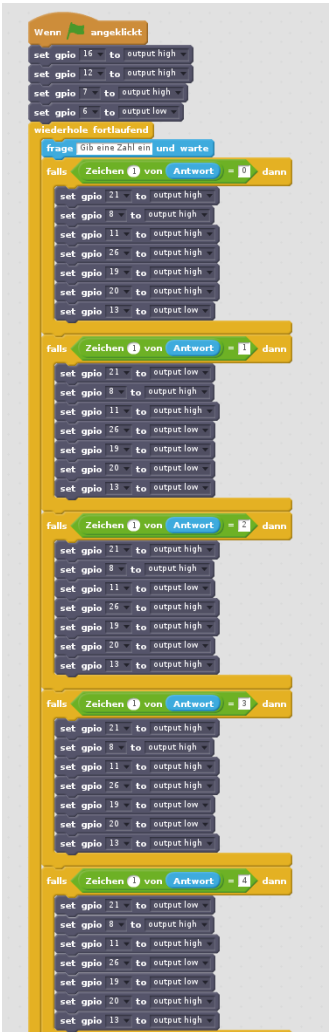
If the first digit is 0, GPIO pins 21, 8, 11, 26, 19, 20 will be switched on and GPIO pin 13 will be set to Low and switched off.

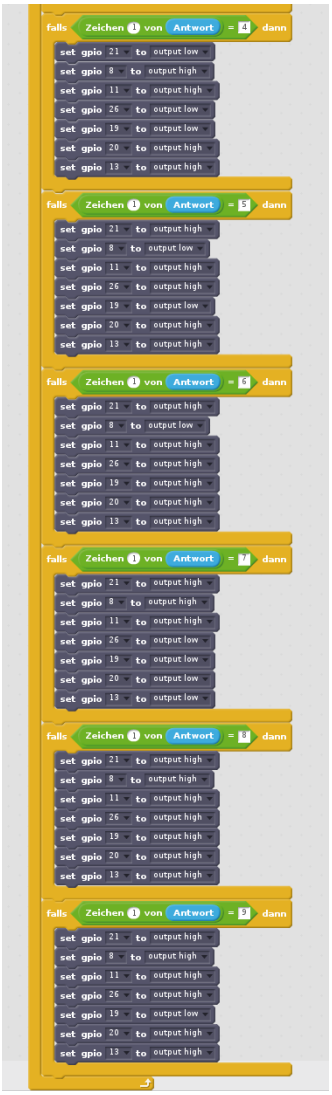
Similar queries will follow for all other digits. All seven segments are always switched, not just the required ones, so that none of the last digits displayed remains.

The Scratch cat asks for a number

Duplicate blocks
When building a Scratch program, you do not need to create similar block combinations every time. Right-click on the first block you want to duplicate. Then select **Duplicate** from the menu. All blocks hanging below are automatically duplicated. The duplicated blocks can then be inserted once more in the appropriate place in the program.

If the user does not enter a digit, but other characters, the last number displayed on the display will not change.





State of the GPIO pins if the user has entered a 0.

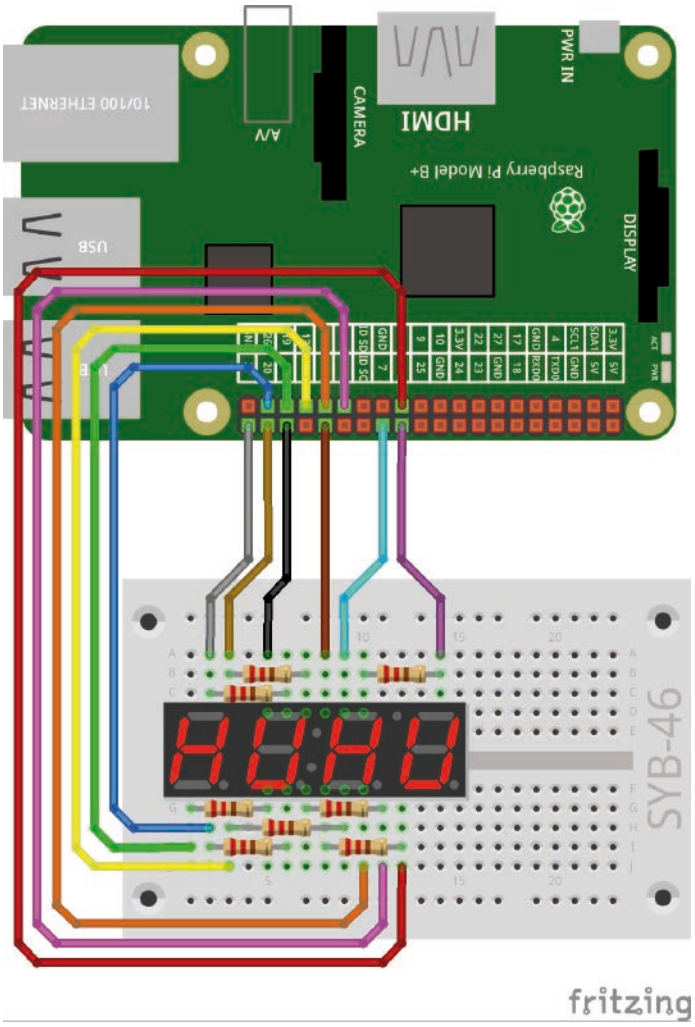
The whole program at a glance

16. Day

16th day

Today in the Advent calendar

- 1 resistor 220 ohms



All seven segments and the decimal point of the seven-segment display are connected. The digits are individually connected to GPIO pins.

Digital clock

In the experiment of the 16th day, we build a working digital clock that displays the current time on the seven-segment display. The digital clock can run automatically on the Raspberry Pi without the monitor and keyboard being connected.

In contrast to a PC, the Raspberry Pi does not have a built-in battery-buffered clock, but always fetches its time from a time server on the Internet, which provides the current time for every time zone in the world at all times with the highest precision.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 8 220-ohm resistors (red-red-brown), 12 GPIO connection cables

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. This time, all segments, including the decimal point, are used.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

The program

The program `16uhr.py` is based on the Python programs of the last days with a few additions.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

number=[
  "abcdef", #0
  "bc", #1
```

```
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]
z = [0,0,0,0]
print("Strg+C ends the program")

def za():
    for i in range(4):
      for s in "abcdefg":
        GPIO.output(seg[s], 0)
      GPIO.output(digit[i], 0)
      for s in number[z[i]]:
        GPIO.output(seg[s], 1)
      if i == 1:
        GPIO.output(dp, 1)
      else:
        GPIO.output(dp, 0)
      time.sleep(0.001)
      GPIO.output(digit[i], 1)

try:
  while True:
    time = time.localtime()
    h = time.tm_hour
    m = time.tm_min
    z[0] = int(h / 10)
    z[1] = h % 10
    z[2] = int(m / 10)
    z[3] = m % 10
    while time.localtime().tm_min == m:
      za()

except KeyboardInterrupt:
  GPIO.cleanup()
```

Decimal point to separate hours and minutes.

How it works

Large parts of the program will seem familiar to you. Most things regarding the initialisation of the GPIO ports and the list for displaying the digits on the seven segments of the display is taken from previous programs. To represent the decimal point between the second and third digit, additions are necessary.

```
dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)
```

After defining the GPIO pins for segments and digits, the GPIO pin `5` is set for the decimal point. Since, like the segments, it is a common anode for all four digits, it is set to `0` at the beginning to deactivate the decimal points.

After the segments of a digit are switched on, within the function `za()`, when the second digit is currently lit, the decimal point is also added.

```
    if i == 1:
      GPIO.output(dp, 1)
    else:
      GPIO.output(dp, 0)
```

If the loop counter is set to `1` - the first digit has the number `0` - the GPIO output for the decimal point is set to `1` and thus switched on. For all other digits it is set to `0` and thus switched off.

An endless loop will then start, which will determine the current time and write the digits into the list `z[]` , to show them with the function `za()`.

```
try:
  while True:
    time = time.localtime()
```

In each run, the current time is written to object `time`, regardless of how long it lasts. The function `time.localtime()` from the `time` library is used for this purpose. The result is a data structure that consists of different single values.

```
    h = time.tm_hour
    m = time.tm_min
```

The two values relevant for the digital clock, hours and minutes, are written from the structure into the variables `h` and `m`.

The following lines determine the individual digits for the display from the time specification and write them to the list `z[]`, from which they are then read by the function `za()`.

```
    z[0] = int(h / 10)
```

The first digit of the display shows the tens digit of the hours. For this purpose, the current hour specification from the variable `h` is divided by 10 and the integer value is determined. The result can be 0, 1 or 2.

```
    z[1] = h % 10
```

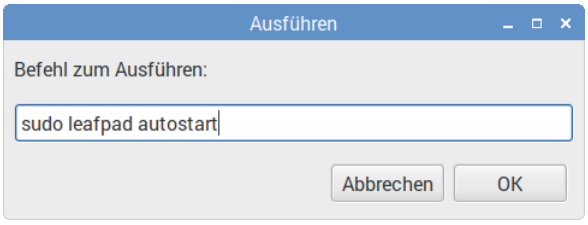
The second digit of the display shows the ones digit of the hours. This is calculated with the modulo operator `%`. Modulo determines the indivisible remainder of an integer division. When dividing by 10, the modulo operator always gives the last digit of the dividend. Since integer values are always calculated, no `int()` function is necessary.

```
    z[2] = int(m / 10)
    z[3] = m % 10
```

In the same way, the third and fourth digits of the list `z[]` are written using the two digits of the current minute specification.

```
    while time.localtime().tm_min == m:
      za()
```

As long as the current minute is still equal to the stored minute, the function `za()` is constantly repeated to output numbers. If the minutes are changed to the next minute, the main loop starts anew, determines the digits of the current time and displays them again until the next minute.



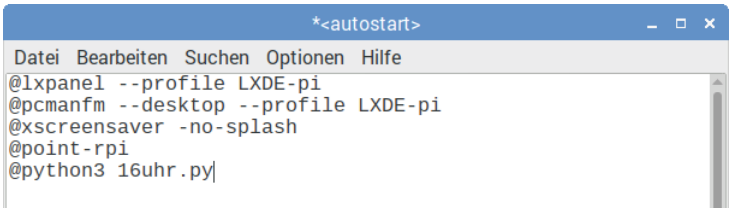
Edit the autostart file with superuser rights

Start digital clock automatically

This program can turn the Raspberry Pi into a digital clock that starts automatically without the Raspberry Pi requiring a keyboard or monitor.

Without keyboard and monitor the program must be started automatically, because the user can call neither Python shell nor command line.

In the file manager, in the directory `/etc/xdg/lxsession/LXDE-pi` open the text file `autostart` in the text editor. Since this file can only be edited with superuser rights, use the file manager to change to the directory and then select **Tools / Execute a command in the current folder in the menu ...** Enter the command sequence shown here.
At the end of the file, add the line shown to call the program with the Python command line interpreter and place an @ character before the line `point-rpi`.



The autostart call for the clock

This automatically displays the current time on the seven-segment display shortly after booting the Raspberry Pi.

To disable the automatic start, delete the line again, or enter a #character at the very beginning of the line:

```
# @python3 16uhr.py
```

The automatically started clock cannot be terminated with the key combination [Ctrl]+[C], since no Python shell window exists.

17th day

Today in the Advent calendar

- 1 GPIO connection cable

The connection diagram of the seven-segment display is the same as on the previous day, although the number on the far right is not used.

Show IP address of Raspberry Pi

The person operating a Raspberry Pi without keyboard and monitor only over the network, requires its IP address in order to establish the SSH connection. This address can be found using network scanner software. It can also be displayed via the network icon on the desktop. It is much more interesting to display the IP address with a seven-segment display directly on the Raspberry Pi.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 8 220-ohm resistors (red-red-brown), 12 GPIO connection cables

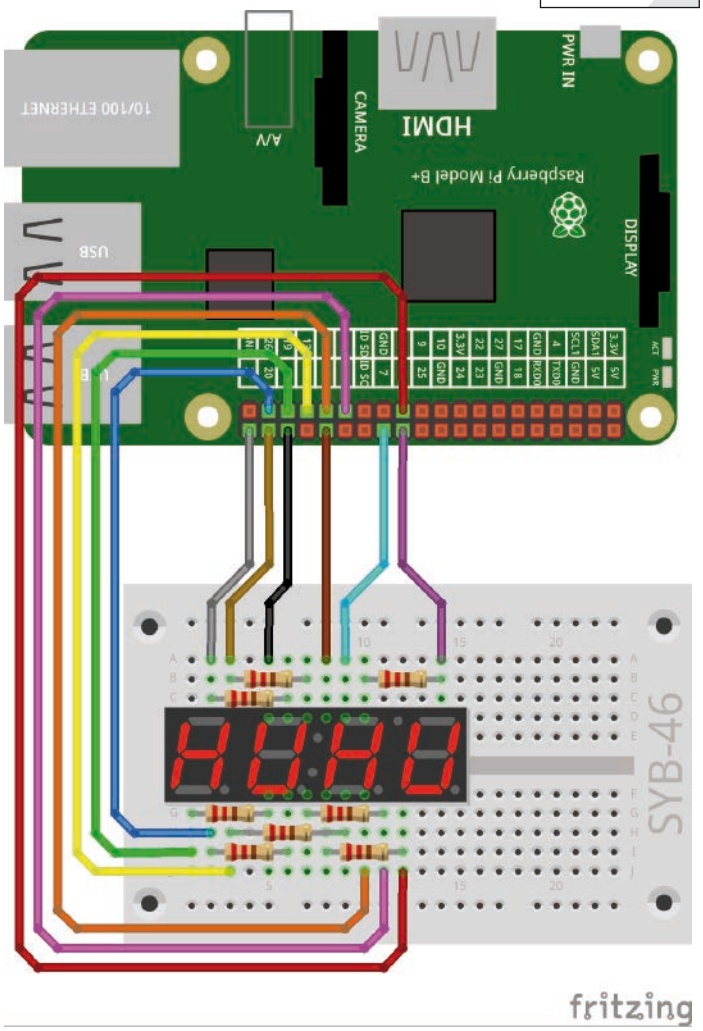
The program

The program `17ip-adresse.py` displays the IP address on the seven-segment display.

The seven-segment display can display four digits simultaneously. However, an IP address consists of four blocks with three digits each. To display these, we display each number block one after the other for about 1 second. To clearly mark the end of the IP address and the beginning of the next display loop, three decimal points of the display should flash briefly after the fourth number block. The display of the IP address then starts again at the first number block.

Number blocks consisting of only one or two digits are also displayed in three digits for a uniform appearance and are supplemented with zeros at the beginning. This spelling is also valid when IP addresses are specified.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, os

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
```



All seven segments and the decimal point of the seven-segment display are connected. The digits are individually connected to GPIO pins.

Display sequence of an IP address on the seven-segment display.

```
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

z = [0,0,0]
ip = os.popen("hostname -I").readline()[:-2].split(".")

print("Ctrl+C ends the program")

def za():
    for i in range(3):
      for s in "abcdefg":
        GPIO.output(seg[s], 0)
      GPIO.output(digit[i], 0)
      for s in number[z[i]]:
        GPIO.output(seg[s], 1)
      time.sleep(0.001)
      GPIO.output(digit[i], 1)

def blink():
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    for k in range(3):
      GPIO.output(digit[k], 0)
    GPIO.output(dp, 1)
    time.sleep(0.5)
    GPIO.output(dp, 0)

try:
  while True:
    for j in ip:
      for k in range(3):
        z[k] = int(j.zfill(3)[k])
      sek = time.time()
      while time.time() <= sec + 1:
        za()
    blink()

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

Large parts of the program will seem familiar to you. Much is based on the earlier Python programs with the seven-segment display. The functions for reading the IP address as well as some methods of string processing are new.

```
import time, os
```

When importing the libraries at the beginning, the module `os` for operating system functions is additionally imported.

```
digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)
```

The list of GPIO ports for the four digits remains unchanged. As the fourth unused position of the display in the program cannot have an undefined status, it is initially initialised and switched off.

```
z = [0,0,0]
```

The list `z[]`, in which the digits to be displayed are stored during program execution, contains only three elements.

```
ip = os.popen("hostname -I").readline()[:-2].split(".")
```

This line first determines the IP address as a character string. The function `os.popen()` executes any command line command. The method `readline()` reads a line as a string from this result.

The four-part IP address is separated at the points into four blocks, which are stored as four individual character strings in the list `ip[]`. The method `split()` is used, which is available in every string. As a parameter, it requires the separator character at which the character string is to be split; in this case the period. This separator itself does not appear in any of the resulting character strings.

For example, if the IP address is

```
"192.168.2.124"
```

it will result in this list `ip[]`:

```
('192', '168', '2', '124')
```

The four elements of the list are still character strings and not numbers.

The function `za()` for number output corresponds to the last programs. Only the lines for displaying the decimal point between the second and third digit of the clock are omitted.

```
def blink():
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    for k in range(3):
      GPIO.output(digit[k], 0)
    GPIO.output(dp, 1)
    time.sleep(0.5)
    GPIO.output(dp, 0)
```

In addition, a function `blink()` is defined, which flashes the three decimal points of the digits used for 0.5 seconds. First all seven segments are set to `0` and thus switched off. The cathodes of the first three digits are then set to `0` and the decimal point to `1`.

Now the three decimal points will light up. After a waiting time of 0.5 seconds, the GPIO pin of the decimal point is reset to `0` , and the LEDs are switched off again. The main program again consists of an endless loop, which cyclically displays the four number blocks of the IP address.

```
    for j in ip:
```

To do this, a loop runs over the four elements of the list `ip[]` in which these number blocks are stored.

```
      for k in range(3):
```

In each block, an inner loop runs three times to represent three digits each. These are stored in the three elements of the list `z[]`, which the function `za()` later reads out.

```
        z[k] = int(j.zfill(3)[k])
```

This line writes the numerical value of a digit into the current element of the list `z[]` in each pass. This is read with the function `int()` from a digit stored as a character.

From the outer loop the counter `j` takes over a numeric block of the IP address. The method `zfill()`, which is available in every character string and has already been used in an earlier program, supplements a character string to a certain length specified in the parameter (here `3`), with the original character string on the right in the result. On the left, any missing areas are filled with zeros. This way, three digits are always stored in the list `z[]`, even if a numeric keypad of the IP address consists of only one or two digits.

Thus, `ip[]` is used from the field as an example above.

```
('192', '168', '2', '124')
```

in four runs, these fields `z[]`:

```
(1, 9, 2)
(1, 6, 8)
(0, 0, 2)
(1, 2, 4)
```

The function `za()` later reads the field `z[]` and displays the three digits on the seven-segment display. A `while` loop repeatedly runs the function `za()` for 1 second to display numbers and then ends the main loop.

```
    sek = time.time()
    while time.time() <= sec + 1:
      za()
```

In the variable `sek` the current time is stored in seconds, which the function `time.time()` outputs at any time to the nearest hundredth of a second. The `while` loop queries whether one or more seconds have passed since the time was saved. As long as this is not the case, the function `za()` is called again and again to display numbers. After the second has lapsed, the outer loop starts again using the next number block of the IP address.

After this loop has passed four times, the function `blink()` is called, which flashes the decimal points once to indicate the end of the IP address. The loop then starts all over again. This is repeated until the user terminates using [Ctrl]+[C]. This program, similar to the 16th day clock program, can be started automatically when the Raspberry Pi is started, to display the IP address, even if no screen is connected.

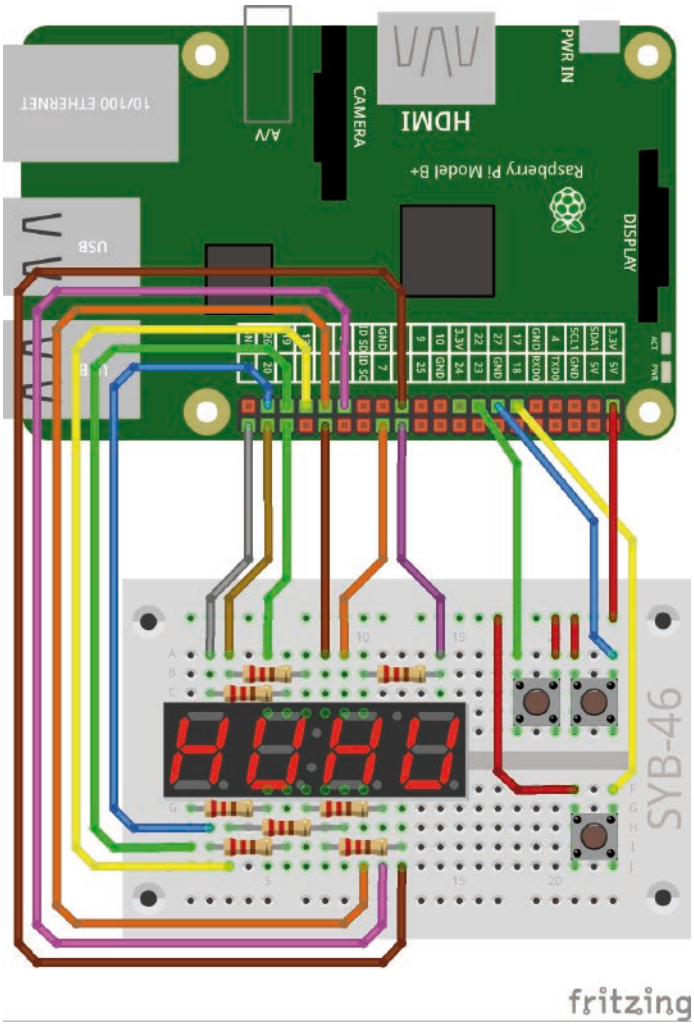
18. Day

18th day

Today in the Advent calendar

- 1 button

The connection diagram of the seven-segment display is the same as for the previous days.



Seven-segment display and three buttons.

Stopwatch

The experiment of the 18th day is a stopwatch with three buttons. The first button starts the stopwatch, the second stops to the nearest tenth of a second and the third resets the measured time to 0.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 8 220-ohm resistors (red-red-brown), 3 buttons, 16 GPIO connection cables, 3 wire jumpers (different lengths)

The program

Program `18stoppuhr.py` runs a stopwatch on the seven-segment display. The watch measures the time in tenths of a second, therefore the decimal point lights up behind the third digit. The buttons are connected to GPIO pins 22, 27 and 17.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

t1=22
t2=27
t3=17
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

print("Start: Button 1")
print("Stop: Button 2")
print("Reset: Button 3")
print("Ctrl+C ends the program")
```

```
def za():
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    if i == 2:
      GPIO.output(dp, 1)
    else:
      GPIO.output(dp, 0)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)

try:
  while True:
    z = [0,0,0,0]
    while GPIO.input(t1)==0:
      za()
    start = time.time()
    while GPIO.input(t2)==0:
      time = time.time() - start
      z[0] = int(time % 1000 / 100)
      z[1] = int(time % 100 / 10)
      z[2] = int(time % 10)
      z[3] = int(time * 10 % 10)
      while int((time.time() - start) * 10 % 10) == z[3]:
        za()
    while GPIO.input(t3)==0:
      za()

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

The program is based on known elements'; the initialisation of the GPIO pins for the seven-segment display is also the same as for the previous days.

```
t1=22
t2=27
t3=17
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)
```

In addition to the GPIO pins of the seven-segment display, three pins are initialised as inputs with pull-down resistors for the buttons.

```
print("Start: Button 1")
print("Stop: Button 2")
print("Reset: Button 3")
print("Ctrl+C ends the program")
```

A short explanation of the buttons appears before the actual start of the program.

```
try:
  while True:
    z = [0,0,0,0]
```

The main loop of the program sets the stopwatch to 0 at the start of each cycle. The four digits in the list `z[]` are set to 0.

```
    while GPIO.input(t1)==0:
      za()
```

As long as button 1 is not pressed, function `za()` displays four zeros on the seven-segment display.

```
if i == 2:
  GPIO.output(dp, 1)
else:
  GPIO.output(dp, 0)
```

This function is slightly changed. Whenever the third digit lights up, the decimal point is also turned on. For the other digits it won't light up.

```
start = time.time()
```

When the user presses button 1, the first `while` loop ends. The current time in seconds is stored in the variable `start`. From this time, the current time of the stopwatch is always calculated, which was started by pressing button 1.

The function `time.time()` specifies the current time as Unix timestamp.

What is the Unix Timestamp?
The Unix timestamp, also known as the Unix epoch, is a time format commonly used in Unix and Linux that specifies the number of seconds that have lapsed since 01.01.1970 00:00 UTC. This is a ten-digit number. Fractions of a second follow the comma. The Unix timestamp on 01.12.2019 00:00 UTC is 1575158400.

```
while GPIO.input(t2)==0:
  time = time.time() - start
  z[0] = int(time % 1000 / 100)
  z[1] = int(time % 100 / 10)
  z[2] = int(time % 10)
  z[3] = int(time * 10 % 10)
```

The next loop runs until the user presses the stop button 2. In the variable `time` the time elapsed since the start of the stop timer is calculated in each run by subtracting the stored start time from the current time.

In the four elements of the list `z[]` whole seconds are calculated and stored as a three-digit number and tenths of a second at the fourth position.

```
  while int((time.time() - start) * 10 % 10) == z[3]:
    za()
```

A loop will now start, displaying along with the function `za()` the digits stored in the list `z[]` until one tenth of a second has lapsed since the last displayed time. Only once a tenth of a second has lapsed the superior `while` loop will start the next run. Now new values are determined for the digits in the list `z[]` - provided the stop key 2 has not yet been pressed.

```
while GPIO.input(t3)==0:
  za()
```

If stop button 2 is pressed, the loop ends and another `while` loop starts waiting for the user to press rest button 3. As long as this is not the case, the last stored digits are displayed in the list `z[]` with the function `za()`.

When the 3 key is pressed, this loop also ends, and the main loop starts its next cycle by first resetting the four digits to 0.

19th day

19. Day

Today in the Advent calendar

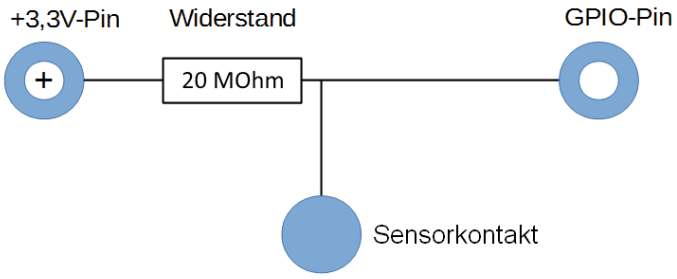
- 1 resistor 20 MOhm (red-black-blue)
- 1 plasticine

The connection diagram of the seven-segment display is the same as for the previous days.

Touch sensor from plasticine

Today traffic lights, door openers, light switches and machines are often controlled with sensor contacts that you only have to touch. Pushbuttons that really need to be pressed are becoming increasingly rare.

The GPIO pin switched as input is connected to +3.3 V via an extremely high-impedance resistor (20 MOhm), so that there is a weak but clearly defined high signal present. A person who is not floating freely in the air is always earthed and delivers a low level current through his electrically conductive skin. If this person touches a sensor contact, the weak high signal is overcome by the significantly stronger low signal of the hand and pulls the GPIO pin to the low signal. The built-in pull-down resistor at the GPIO input must be switched off.



Sensor contact at a GPIO input

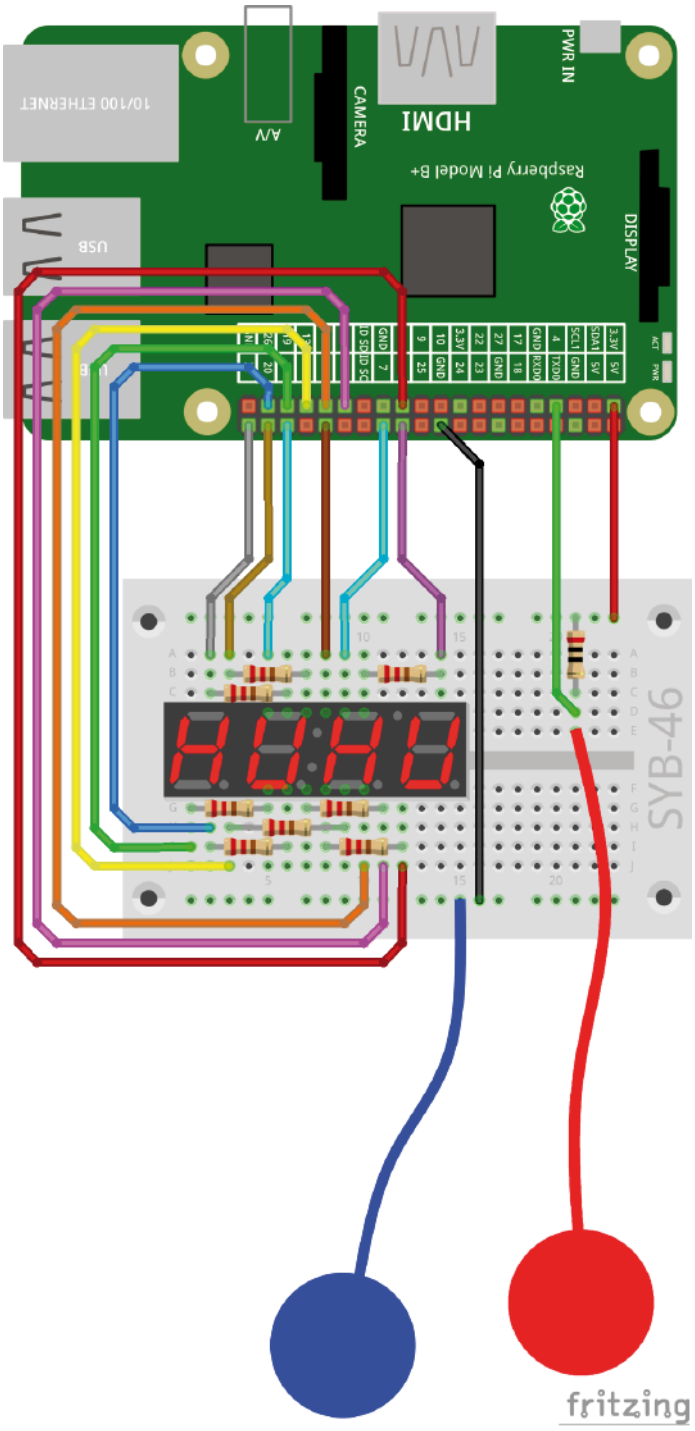
The level of resistance between the hand and the other material depends on many things, including shoes and floors. Barefoot in wet grass is the best connection to the earth's ground, but it also works well on stone floors. Wooden floors insulate more strongly, plastic floor coverings are often even positively charged. If the sensor contact does not work, there is another plasticine contact for the circuit which can be connected to the ground rail of the plug-in board. If you touch this and the actual sensor at the same time, the ground connection is established in any case.

Plasticine conducts electricity almost as well as human skin. It is easily moulded into any shape, and a plasticine contact is much easier to handle than a simple piece of wire. The surface with which the hand touches the contact, is clearly larger. So it is not so easy as with a "loose contact". Insert a piece of stripped wire into a piece of plasticine. Insert the other end of the wire into the clipboard.

Stopwatch with sensor contact

The experiment of the 19th day controls a stopwatch via a simple sensor contact.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 8 220-ohm resistors (red-red-brown), 1 20-MOhm resistor (red-black-blue), 2 plasticine contacts, 15 GPIO connection cables



Seven-segment display and sensor contacts.

The program

The program `19stoppuhr.py` is similar to the program of the 18th day, with the difference that only one sensor contact is used for start, stop and reset.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

k1=4
GPIO.setup(k1, GPIO.IN)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

print("Start, Stop, Reset: Touch sensor contact")
print("Ctrl+C ends the program")

def za():
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    if i == 2:
      GPIO.output(dp, 1)
    else:
      GPIO.output(dp, 0)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)

try:
  while True:
    z = [0,0,0,0]
    while GPIO.input(k1)==1:
      za()
    while GPIO.input(k1)==0:
      pass
    start = time.time()
    while GPIO.input(k1)==1:
      time = time.time() - start
      z[0] = int(time % 1000 / 100)
```

```
      z[1] = int(time % 100 / 10)
      z[2] = int(time % 10)
      z[3] = int(time * 10 % 10)
      while int((time.time() - start) * 10 % 10) == z[3]:
        za()
    while GPIO.input(k1)==0:
      pass
    while GPIO.input(k1)==1:
      za()
    while GPIO.input(k1)==0:
      pass

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

The program is based on known elements; the initialisation of the GPIO pins for the seven-segment display is also the same as for the previous days.

```
k1=4
GPIO.setup(k1, GPIO.IN)
```

In addition to the GPIO pins of the seven-segment display, a pin is initialised as input without pull-down resistor for the sensor contact.

```
try:
  while True:
    z = [0,0,0,0]
    while GPIO.input(k1)==1:
      za()
```

Since a sensor contact jumps to low level when touched, this time the first `while` loop runs within the infinite loop as long as the input `k1` delivers high level.

Since this program version does not use three different buttons for start, stop and reset, but only one sensor, it must be prevented that a longer touch triggers several actions in succession.

```
    while GPIO.input(k1)==0:
      pass
```

A further loop runs as long as the input `k1` delivers low level, i.e. the sensor contact is touched. Nothing happens inside the loop. Python uses the `pass` statement, which simply does nothing. This makes it easy to implement such holding patterns. The next action in the program only takes place after the user has released the sensor contact again. As a conspicuous sign, the display switches off when the sensor contact is touched, as it only lights up while function `za()` is running.

Once the user has released the sensor contact again, the program continues to run, and the stopwatch starts to run.

```
    while GPIO.input(k1)==1:
      time = time.time() - start
```

Here too a `while` loop runs as long as the input `k1` delivers high level, i.e. the sensor contact is not touched. After this loop and also after the reset to 0, the program will wait each time until the user releases the sensor contact.

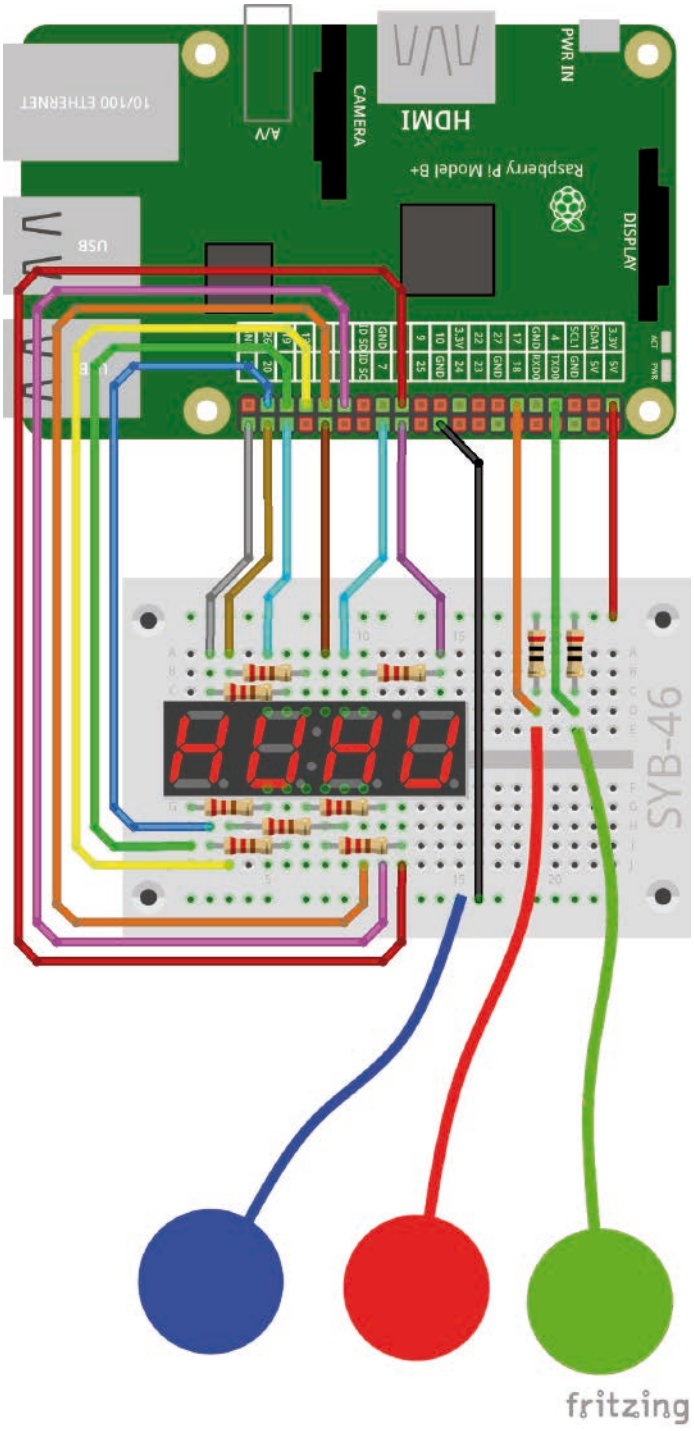
20. Day

20th day

Today in the Advent calendar

- 1 resistor 20 MOhm (red-black-blue)

The connection diagram of the seven-segment display is the same as for the previous days.



Seven-segment display and sensor contacts.

Counter with sensor contacts

The experiment of the 20th day is a simple counter on the seven-segment display controlled by two sensor contacts. One sensor contact counts up; the other counts down. The sensor contacts are connected to GPIO pins 17 and 4.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 8 220-ohm resistors (red-red-brown), 2 20-MOhm resistors (red-black-blue), 3 plasticine contacts, 16 GPIO connection cables

The program

The program `20zaehler.py` counts from 0 upwards to a maximum of 9999 or downwards again to 0 on the seven-segment display. This time there is no waiting until the user releases the sensor contact again. The counting should continue automatically after a longer touch.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

k1=17
k2=4
GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

z=[0,0,0,0]
x=0

print("Sensor contact 1 increases the number")
print("Sensor contact 2 reduces the number")
print("Ctrl+C terminates the program")
```

```
def za(n):
  z[0] = int(n / 1000)
  z[1] = int(n % 1000 / 100)
  z[2] = int(n % 100 / 10)
  z[3] = int(n % 10)
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)

try:
  while True:
    za(x)
    if GPIO.input(k1)==0 and x>0:
      x-=1
    if GPIO.input(k2)==0 and x<9999:
      x+=1

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

This program is also largely based on earlier programs. The functions for initialising the seven-segment display are the same.

```
k1=17
k2=4
GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)
```

For the two sensor contacts, two GPIO pins are initialised as inputs without pull-down resistors.

```
x=0
```

A new variable x is set to 0 in the definitions. Later in the program, it contains the value of the counter that is displayed.

In this program, the calculation of the four digits is stored in the function `za()` , which simplifies the main program. This function contains a parameter that is passed when the function is called.

```
def za(n):
  z[0] = int(n / 1000)
  z[1] = int(n % 1000 / 100)
  z[2] = int(n % 100 / 10)
  z[3] = int(n % 10)
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)
```

The parameter `n` in the function definition is a four-digit number that will be displayed. The first four lines of the function determine the individual digits from the number. The function then runs as in previous programs and displays these four digits on the seven-segment display.

```
try:
  while True:
    za(x)
```

The main loop of the program displays the current value of the counter `x` on the seven-segment display in each pass via function `za(x)`.

```
    if GPIO.input(k1)==0 and x>0:
      x-=1
```

If the sensor contact `k1` is touched and the counter is still greater than 0, it is reduced by 1. This ensures that the counter does not reach any negative values.

```
    if GPIO.input(k2)==0 and x<9999:
      x+=1
```

If the sensor contact `k1` is touched and the counter is still smaller than 9999, it is increased by 1. This ensures that the counter does not reach five-digit values that can no longer be displayed.

The main loop then restarts and displays the current value of the counter again.

The counter counts too fast.
If you try the program, you will notice that the counter already reacts to the shortest touch and it is very difficult to count individual steps.

Of course, as with the previous program, you could wait for the user to release the sensor contact every time. But then it would not be possible to quickly count quickly by touching for a long time.

The program `20zaehler02.py` contains a short waiting time after each change of the counter, in order to also count individual steps with short touch.

```
try:
  while True:
    za(x)
    if GPIO.input(k1)==0 and x>0:
      x-=1
      time.sleep(0.04)
    if GPIO.input(k2)==0 and x<9999:
      x+=1
      time.sleep(0.04)
```

Change these waiting times as required to be able to count easily.

21th day

21. Day

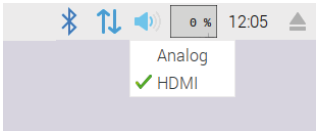
Today in the Advent calendar

· Download code

The connection diagram of the seven-segment display is the same as for the previous days.

The Raspberry Pi produces tones

The Raspberry Pi can play music through an HDMI monitor, external speaker or headphones on the 3.5 mm analogue jack. For computer monitors with a DVI connector connected to the HDMI output, a speaker must be connected to the analogue output because the audio signal is not transmitted via the DVI cable. Right-click the speaker icon in the upper right corner to select the audio output.



Select audio output

Components: 1 plug-in board SYB-46, 1 seven-segment display, 8 220-ohm resistors (red-red-brown), 12 GPIO connection cables

The program

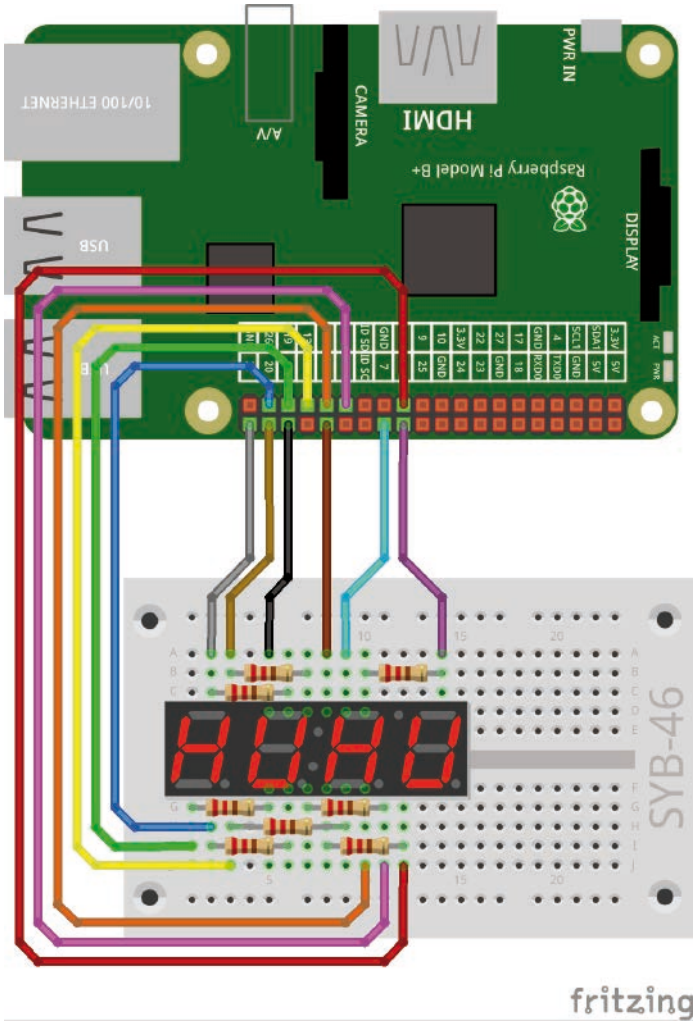
The Scratch-2 program `21musik.sb2` displays a simple keyboard. When you click a button, you will hear the corresponding sound and the seven-segment display will display the sound as a letter.

Start a new project in Scratch and first delete the cat. In this project, we need various other objects, more precisely, one object for each keyboard key, but not the cat.

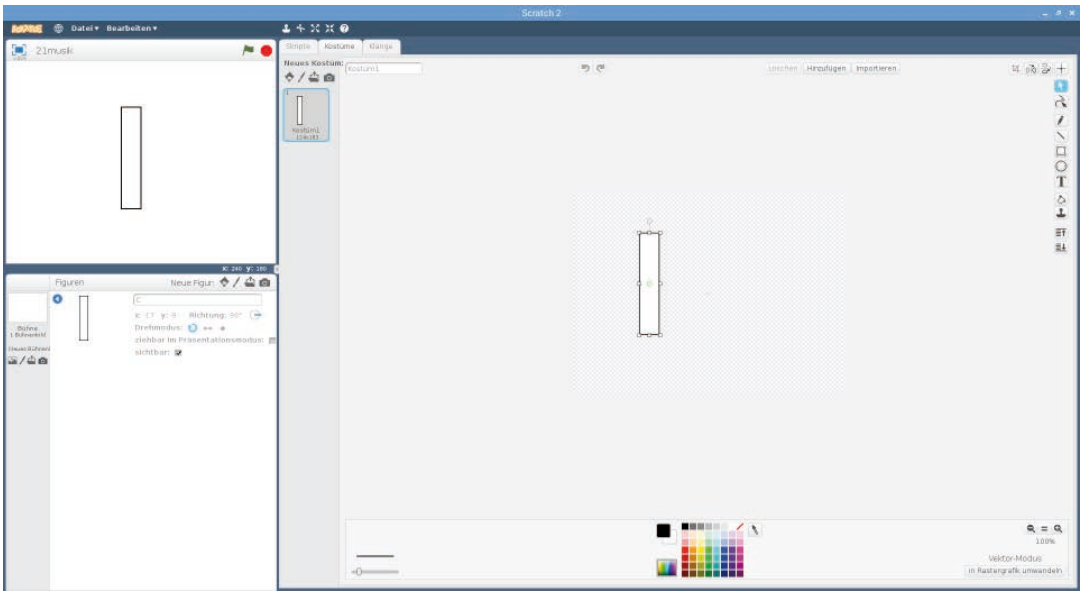
Draw the first white keyboard key. To do this, switch to vector graphics in the graphics area at the bottom right. Draw a narrow unfilled black rectangle.

Then select the white colour, click on the colour bucket icon and fill in the rectangle.

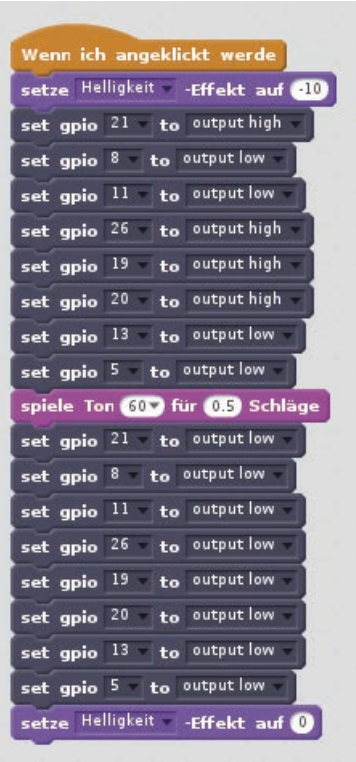
Give the key another name. With all the many objects in a program, one can easily get confused. Click on the blue i symbol of this key in the figure list and change the name in the name field to `c`. This is the tone that should sound when clicking the first key.



Seven-segment display without additional components.



The first keyboard key



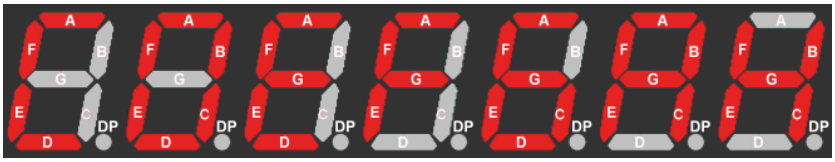
The scratch blocks for the C key

Each key now gets its own program blocks. These are similar for all keys and play a specific tone when the key is clicked. If you are assembling the program for the first key, you can duplicate it for the other keys and only need to change it slightly.

The program blocks do not start when the green flag is clicked, but each time the corresponding key is clicked. Select the key **C** in the figure list and drag **When I am clicked** into the script window from the block palette **Events**.

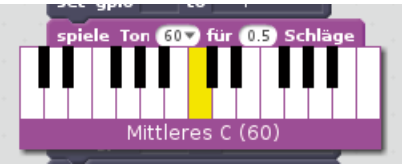
All blocks below are always executed when the figure - here the key **C** - is clicked. To see that a key is clicked, it should appear a little darker for a moment. To do this, from the block palette **Appearance** append the block ...- **Effect to** ... to the program, select in the list box **Brightness** and enter in the number field `-10`. The figure will darken.

The seven segments of the seven-segment display are then switched to display the name of the sound as letters. Seven-segment displays are only conditionally suitable for displaying letters. But with a little imagination you can recognise the letters needed for the keyboard.



The letters of the tones used on the seven-segment display

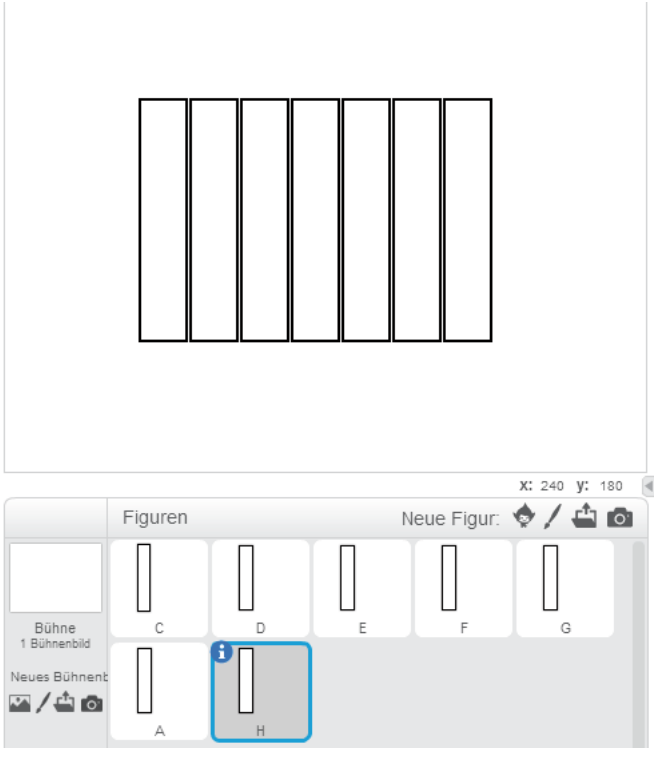
To create the right sound, build the block **play sound... for... beats** from the block palette **Sound** into the program.



Keyboard for the selection of tones

Simply click once on this block in the program window and the first tone will sound. Then click in the number field behind **tone**. A keyboard appears on which you can select the desired tone. The first piano key **C** will play the tone **C(60)**.

All seven segments of the seven-segment display are then switched off. The key should also return back to the normal display and not be darkened. To do this, hang another block **set brightness effect on...** to the program and set the value back to `0`.



The white keys of the keyboard

Our keyboard should have seven white keys and five black keys. You can easily duplicate the next six white keys from the first one. To do this, right-click on the figure `C` in the figure list and select Duplicate **from the menu**.

Rename the duplicated button to `D` and move it a tiny distance to the right of the first button. Duplicate other buttons in the same way and call them `E`, `F`, `G`, `A` and `B`.

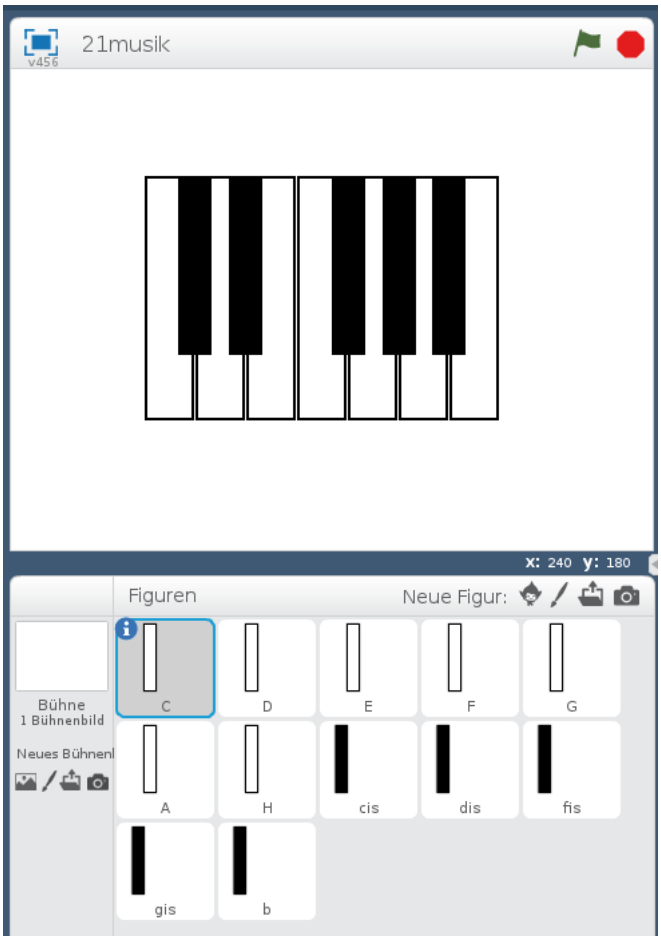
After duplicating, change the blocks for all keys to display the segments in such a way that the respective letter is displayed. Also select the appropriate tones for the buttons. Which key should play which tone is clearly visible in the piano key image of the block **play sound ... for ... beats**. The names of the sounds in Scratch sometimes differ slightly from the German names.

German name	Key colour	Number	Scratch
c	white	60	Middle C
cis	black	61	C#
d	white	62	D
dis	black	63	Eb
e	white	64	E
f	white	65	F
fis	black	66	F#
g	white	67	G
gis	black	68	G#
a	white	69	A
b	black	70	Bb
h	white	71	B

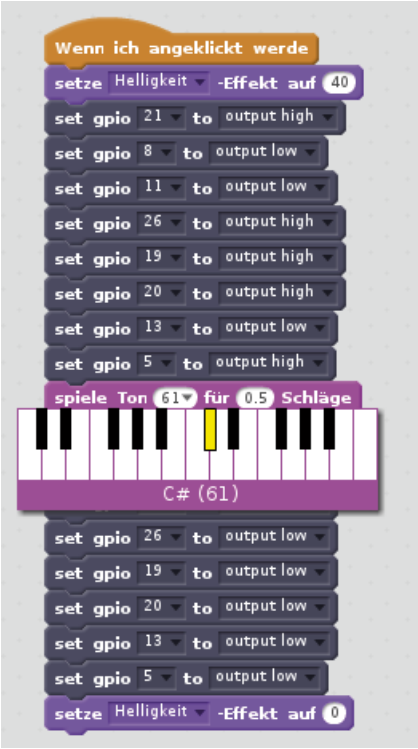
Draw the first black key as a filled rectangle, slightly smaller than the white keys. Also use the mode **Vector graphic**. Slide this button between the first two white buttons so that the top edges are flush with each other.

The program blocks of the black keys are largely the same as those of the white keys. The black keyboard keys must become a little brighter and not darker when you click on them, so that they are easy to recognise. Set the brightness to `40` instead of `-10`. As letters for the sounds, we use the letter of the white key to the left of it and turn on the decimal point for differentiation.

Duplicate four more black keys. Insert them as shown between the white buttons and call them `D#`, `F#`, `D#` and `Bb`.



The white and black keys of the keyboard



The Scratch blocks for the black C# key

Then adjust the program blocks of the keys accordingly. Now you can play music on the keyboard.

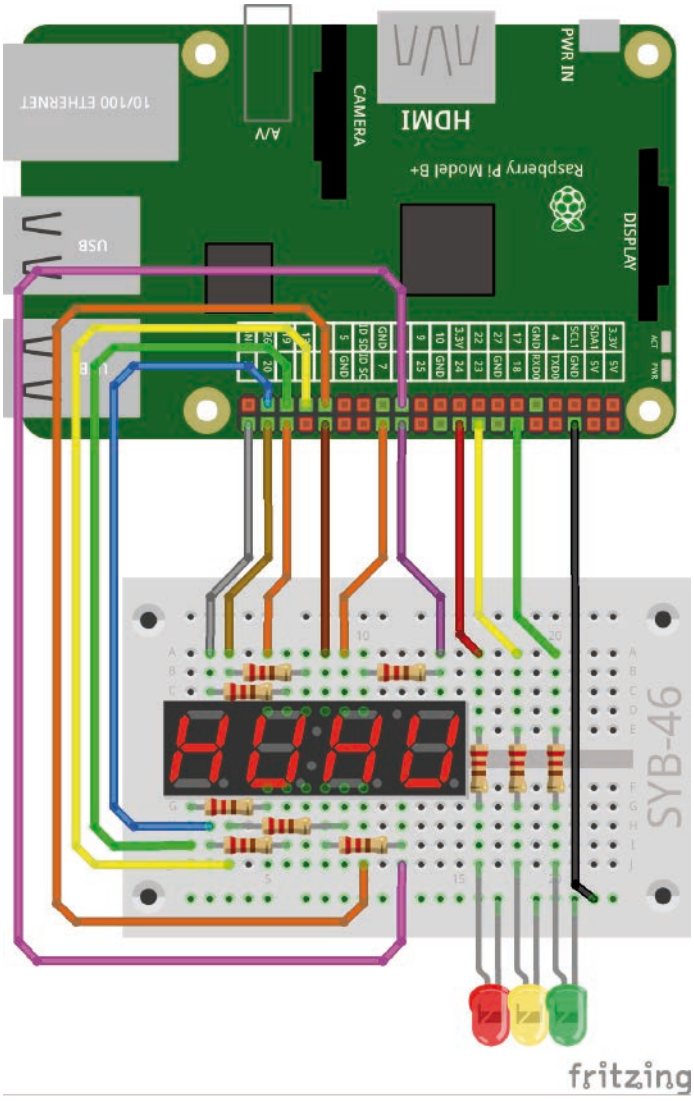
You can download sheet music for a Christmas song using the download code in the Advent calendar today.

22. Day

22th day

Today in the Advent calendar

- 2 resistors 220 ohms



Seven-segment display and three LEDs.

The connection diagram of the seven-segment display is the same as for the previous days. The decimal point is not used.

Traffic light with countdown

The experiment of the 22nd day simulates a traffic light with three LEDs. During the green phase, a countdown shows how much time you have left.

Components: 1 plug-in board SYB-46, 1 seven-segment display1 LED red, 1 LED yellow, 1 LED green, 10 220-Ohm resistors (red-red-brown), 15 GPIO connection cables

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pin is not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

The LEDs of the traffic light are connected to the GPIO pins 24, 23 and 18.

The program

The Scratch-2 program `22ampel.sb2` runs the traffic light cycle by clicking on the green flag. A self-defined block is used for the countdown. This can also be used at any time in other programs to display a digit.

The beginning of the program contains known elements. GPIO pins are initialised one after the other:

- The pins 16, 12, 7 for the cathodes of the first three digits are set high and these digits are turned off, the cathode of the fourth digit on pin 6 is set to low to turn on this digit.
- The anodes of the seven segments: 21, 8, 11, 26, 19, 20, 13 are set to Low and all segments are switched off.
- The red LED at pin 24 is then switched on, the yellow and green LED at pin 23 and are switched off.
- After half a second, the yellow LED at pin 23 is also switched on.
- After another half second, the red and yellow LEDs are switched off and the green LED at pin 18 is switched on.

After that, it gets interesting.

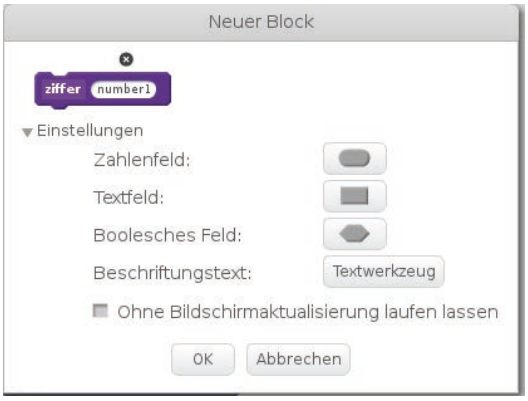
A variable z is counted down in a countdown loop. It contains the number to be displayed in each run.

Variables in Scratch

Variables are small memory locations where a program remembers a number or something else. When the program is closed, these variable memories are automatically emptied. Variables have to be created in Scratch on the block palette **Data** by clicking on **New variable** before they can be used. You can then drag the symbol of the newly created variable from the block palette to a designated field of a block in the program. The block palette also contains various blocks for reading and changing variables.

Create a variable **z** for the counter.

The new variable z



Defining new block

For the representation of a digit on the seven-segment display, we use our own, self-made block. Actually this would not be necessary for this program, in which each digit is represented only once. However, the program shows how you can use this technique in other programs. Self-built blocks in Scratch are comparable to functions in Python that can be defined once and then called again.

Click on the button **New Block** on the block palette **Additional Blocks** . Every new block needs a name. In the example we call the block digit. Put a number field in this block. It should later contain the **digit** that the block will be displaying on the seven-segment display.

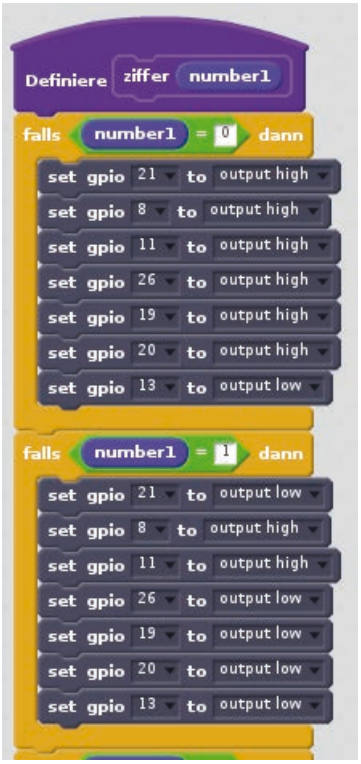
After clicking **OK** a new block **define digit number1** will appear, which is round at the top, so it is always the beginning of a sequence of blocks. The blue block **number1** later contains the number that is given to the block when it is called.

Hang ten **if ... then** queries under this block that check whether the number **number1** equals 0, 1, 2, 3, etc.. In each case, the seven segments of the seven-segment display are switched so that they represent the corresponding digit. The blocks for switching the segments correspond to the program of the 15th day.

You can simply insert the block **number1** from the block definition into the fields of the ... = ... query blocks.

A block **Additional blocks** a block named **digit** ...will appear. You can use this like any standard block in the program.

Initialisation of the GPIO pins



Defining self-built block

The new block on the block pallet

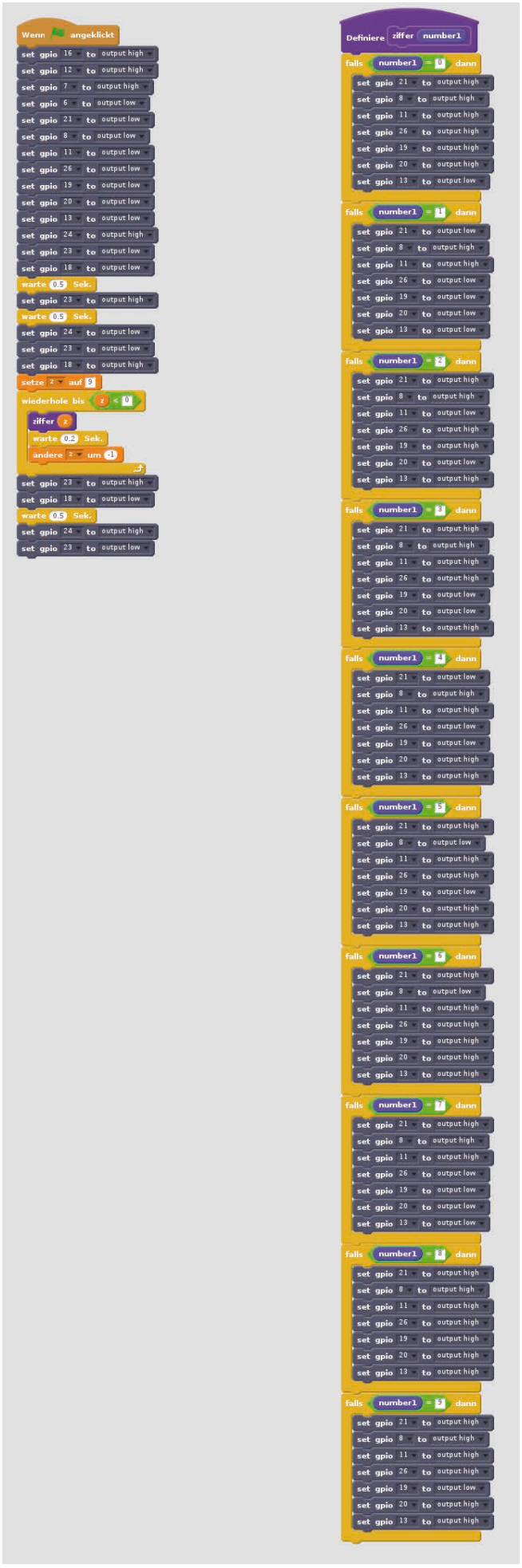
In the main program, the countdown counter z is first set to 9 after the green LED of the traffic light has been turned on. A loop then starts, which runs until the counter z is smaller than 0.

In each loop pass, the block digit ... is generated and the with the current number z of the countdown so that this number is displayed. After a waiting time of 0.2 seconds, the variable z is counted down by 1 and the loop starts the next pass.

The counting loop in the program

After the countdown has reached 0, the yellow LED of the traffic light is switched on and the green LED is switched off. After a further waiting period of 0.5 seconds, the red LED of the traffic light is switched on and the yellow LED is switched off.

The program is now finished and can be restarted by clicking on the green flag.



The whole program at a glance

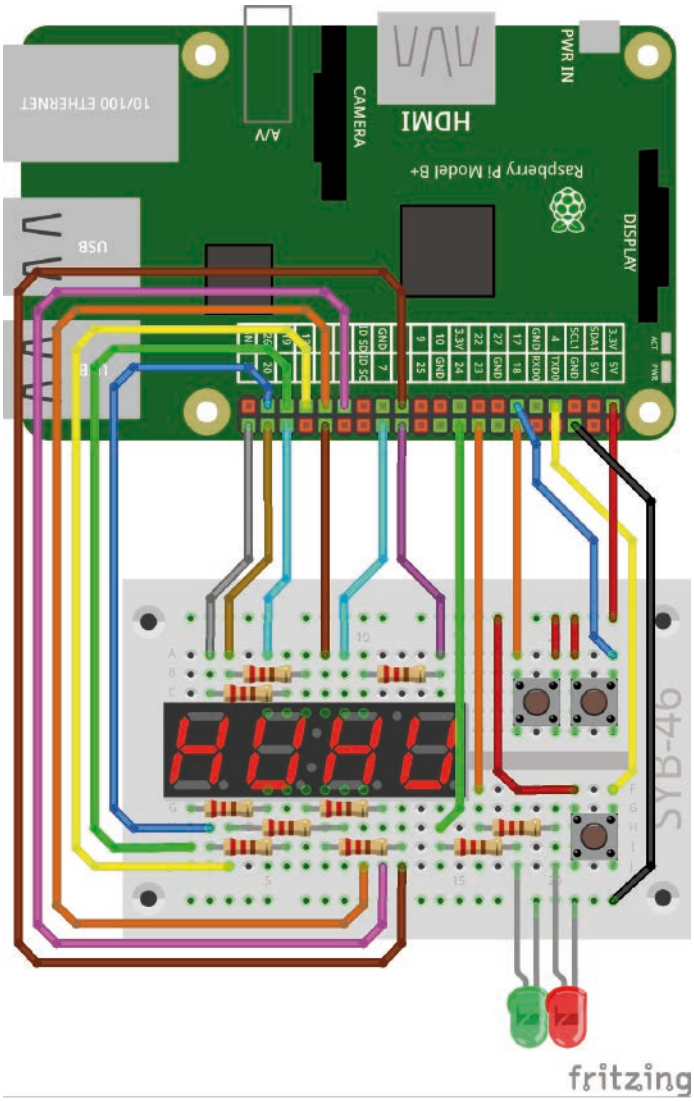
23. Day

23th day

Today in the Advent calendar

· 3 GPIO connection cable

The connection diagram of the seven-segment display is the same as for the previous days. The decimal point is also used.



Seven-segment display, two LEDs and three buttons.

Number rates with three keys

This is a simple guessing game in which a random number chosen by the computer should be guessed by the player in as few steps as possible. The LEDs are connected to the GPIO pins 24 and 23, the buttons to the GPIO pins 18, 17 and 4.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 1 LED red, 1 LED green, 9 220-ohm resistors (red-red-brown), 3 buttons, 18 GPIO connection cables, 3 wire jumpers (different lengths)

The LEDs are connected to the GPIO pins 23 and 24, the buttons to the GPIO pins 18, 17 and 4.

The program

The program `23spiel.py` generates a random number between 0 and 100, which the player has to guess with as few tips as possible. With two buttons the player sets a tip and delivers it with the third button.

The two left digits of the display show the typed number, the two right digits show how many attempts the player has already made. The two LEDs indicate whether the number searched for is smaller or larger than the last tip.

How are random numbers generated?

It is commonly thought that nothing can happen randomly in a program - so how can a program be able to generate random numbers? If you divide a large prime number by any value, you get numbers from the umpteenth decimal place that are very difficult to predict. If you increase the divisor regularly the number also change randomly. Although the result may seem random, it can be reproduced at any time by an identical program or by running the same program several times. But if we take a number assembled from some of these digits and divide it again by a number that results from the current seconds of time or the contents of any memory location of the computer, we get a result that cannot be reproduced and is therefore called a random number.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, random

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)
```

```
LED1=24
GPIO.setup(LED1, GPIO.OUT, initial=0)
LED2=23
GPIO.setup(LED2, GPIO.OUT, initial=0)

t1=18
t2=17
t3=4
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

z=[0,0,0,0]
x=0

print("Button 1 reduces the tip")
print("Button 2 increases the tip")
print ("Button 3 returns the tip")
print("Ctrl+C ends the program")

def za(n):
  z[0] = int(n / 1000)
  z[1] = int(n % 1000 / 100)
  z[2] = int(n % 100 / 10)
  z[3] = int(n % 10)
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in zahl[z[i]]:
      GPIO.output(seg[s], 1)
    if i == 1:
      GPIO.output(dp, 1)
    else:
      GPIO.output(dp, 0)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)

try:
  while True:
    y=random.randrange(100)
    t=0
    while True:
      za(x * 100 + t)
      if GPIO.input(t1)==1 and x>0:
        x-=1
        time.sleep(0.08)
      if GPIO.input(t2)==1 and x<99:
        x+=1
        time.sleep(0.08)
      if GPIO.input(t3)==1:
        t+=1
        if y<x:
          GPIO.output(LED1, 1)
          time.sleep(0.5)
          GPIO.output(LED1, 0)
        if y>x:
          GPIO.output(LED2, 1)
          time.sleep(0.5)
          GPIO.output(LED2, 0)
        if y==x:
          for i in range(5):
            GPIO.output(LED1, 1)
            GPIO.output(LED2, 1)
            for j in range(100):
              za(x * 100 + t)
            GPIO.output(LED1, 0)
            GPIO.output(LED2, 0)
            for j in range(100):
              za(x * 100 + t)
          break

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

At the start, the `random` library is imported, which contains functions for generating random numbers. The GPIO pins for the seven-segment display are initialised according to a known scheme. The decimal point is used to visually separate the two digits to the left from the two digits to the right. In addition, GPIO outputs for the two LEDs and inputs with pull-down resistors for the three push-buttons are set up.

```
x=0

print("Button 1 reduces the tip")
print("Button 2 increases the tip")
print ("Button 3 returns the tip")
print("Ctrl+C terminates the program")
```

Before the program starts, the variable `x`, which later contains the number typed by the player, is set to 0. A short explanation of the keys will then appear on the screen.

```
def za(n):
  z[0] = int(n / 1000)
  z[1] = int(n % 1000 / 100)
  z[2] = int(n % 100 / 10)
  z[3] = int(n % 10)
  for i in range(4):
    for s in "abcdefg":
      GPIO.output(seg[s], 0)
    GPIO.output(digit[i], 0)
    for s in number[z[i]]:
      GPIO.output(seg[s], 1)
    if i == 1:
      GPIO.output(dp, 1)
    else:
      GPIO.output(dp, 0)
    time.sleep(0.001)
    GPIO.output(digit[i], 1)
```

The function `za(n)` for displaying a four-digit number has a similar structure to the program of the 20th day. In addition, the decimal point behind the second digit lights up.

```
try:
  while True:
    y=random.randrange(100)
    t=0
```

The main program is an endless loop in which an integer random number smaller than 100 is stored in the variable `y` at the beginning of each cycle. The player must guess this number. In addition, the variable `t` is set to 0, in which the number of tips is later stored.

```
    while True:
      za(x * 100 + t)
```

Within the loop running once through each game, there is also another endless loop. This displays the last set number and the number of tips in each pass. To do this, multiply the number `x` by 100 and add the tips in the variable `t`. The two-digit number now appears before the decimal point, followed by the number of attempts the player has already made.

```
      if GPIO.input(t1)==1 and x>0:
        x-=1
        time.sleep(0.08)
```

If the user presses key 1 and the set number is still greater than 0, the number is reduced by 1. The program then waits 0.08 seconds to intercept multiple actions caused by accidentally pressing the key too long. If necessary, you can adjust this waiting time.

```
      if GPIO.input(t2)==1 and x<99:
        x+=1
        time.sleep(0.08)
```

In the same way, pressing the second key increases the number if it is less than 99.

```
      if GPIO.input(t3)==1:
        t+=1
```

If the player presses the third button, the number of bets is first increased by 1. Then there are three possibilities:

```
        if y<x:
          GPIO.output(LED1, 1)
          time.sleep(0.5)
          GPIO.output(LED1, 0)
```

- If the searched number `y` is smaller than the given tip, both left LEDs illuminate for half a second.

```
if y>x:
  GPIO.output(LED2, 1)
  time.sleep(0.5)
  GPIO.output(LED2, 0)
```

- If the searched number `y` is greater than the given tip, both right LEDs illuminate for half a second.

```
if y==x:
  for i in range(5):
    GPIO.output(LED1, 1)
    GPIO.output(LED2, 1)
    for j in range(100):
      za(x * 100 + t)
    GPIO.output(LED1, 0)
    GPIO.output(LED2, 0)
    for j in range(100):
      za(x * 100 + t)
```

- If the searched number `y` and the given tip are the same, a loop causes both LEDs to flash five times. During the time that the LEDs are lit or switched off, the function `za(n)` is not simply waited for, but called one hundred times in a row to display the correctly typed number and the number of tips. 100 calls of this function take about 0.1 seconds.

```
  break
```

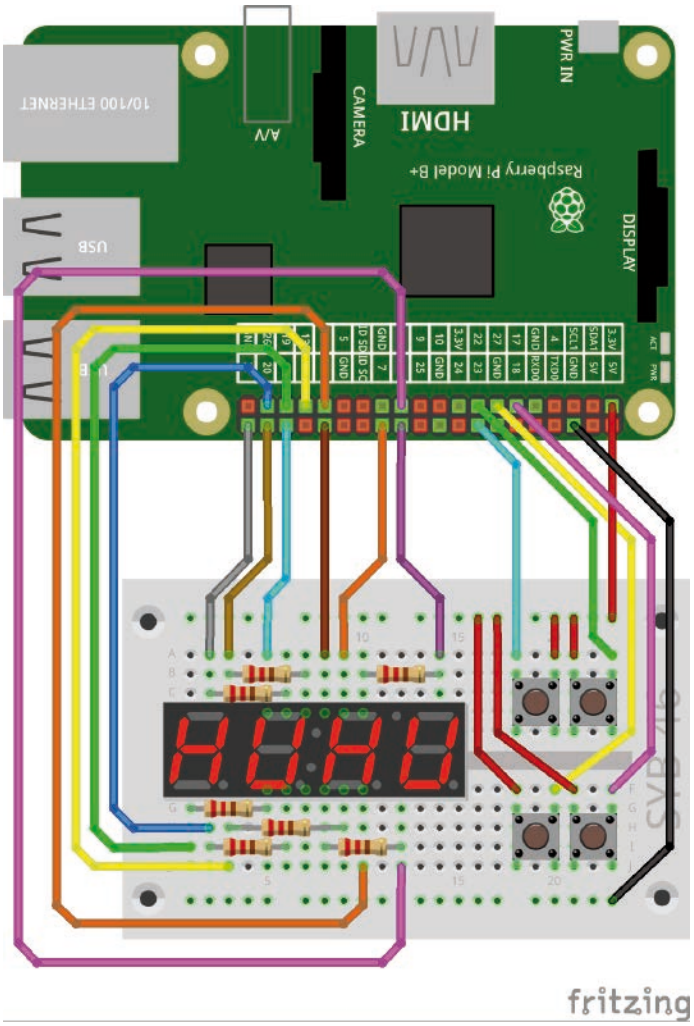
Then, the inner loop of the two loops is aborted because the game has been solved. The outer loop starts a new game, where the last tip is immediately used as the start for the next pass. You therefore don't have to count up from 0 to the desired number anymore.

24. Day

24th day

Today in the Advent calendar

- 1 button
- Download code



Seven-segment display and four buttons.

The connection diagram of the seven-segment display is the same as for the previous days. The decimal point is not used.

Christmas carols on the Raspberry Pi

The experiment of the 24th day plays a Christmas carol when a button is pressed. There are four different carols to choose from. You can download Christmas carols in mp3 format using the download code in the Advent calendar today. The buttons are connected to GPIO pins 22, 27 and 17.

Components: 1 plug-in board SYB-46, 1 seven-segment display, 7 220-ohm resistors (red-red-brown), 4 buttons, 18 GPIO connection cables, 4 wire jumpers (different lengths)

Wiring diagram of the seven-segment display

The following table shows which pins of the seven-segment display are connected to which GPIO pins. The grey pin is not used in this program.

Pin seven-segment display	Pinboard	Segment / digit	GPIO pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

The program

The program `24lieder.py` plays four different Christmas carols, depending on which button the user presses.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, subprocess

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
  GPIO.setup(seg[s], GPIO.OUT, initial=0)

digit=[16, 12, 7, 6]
for z in digit:
  GPIO.setup(z, GPIO.OUT, initial=1)

t1=23
t2=22
```

```
t3=27
t4=17
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t4, GPIO.IN, GPIO.PUD_DOWN)

number=[
  "abcdef", #0
  "bc", #1
  "abdeg", #2
  "abcdg", #3
  "bcfg", #4
  "acdfg", #5
  "acdefg," #6
  "abc", #7
  "abcdefg", #8
  "abcdfg", #9
    ]

print("Ctrl+C ends the program")

def za(n):
  for s in "abcdefg":
    GPIO.output(seg[s], 0)
  for i in range(4):
    GPIO.output(digit[i], 1)
  if n!=0:
    for s in number[n]:
      GPIO.output(seg[s], 1)
    GPIO.output(digit[n-1], 0)

try:
  while True:
    if GPIO.input(t1)==1:
      za(1)
      subprocess.Popen(["omxplayer", "lied1.mp3"])
      time.sleep(4)
      za(0)
    if GPIO.input(t2)==1:
      za(2)
      subprocess.Popen(["omxplayer", "lied2.mp3"])
      time.sleep(4)
      za(0)
    if GPIO.input(t3)==1:
      za(3)
      subprocess.Popen(["omxplayer", "lied3.mp3"])
      time.sleep(4)
      za(0)
    if GPIO.input(t4)==1:
      za(4)
      subprocess.Popen(["omxplayer", "lied4.mp3"])
      time.sleep(4)
      za(0)

except KeyboardInterrupt:
  GPIO.cleanup()
```

How it works

In addition to the already known libraries, the library `subprocess` is imported. This makes it possible to start Linux command line programs from a Python program.

```
def za(n):
  for s in "abcdefg":
    GPIO.output(seg[s], 0)
  for i in range(4):
    GPIO.output(digit[i], 1)
  if n!=0:
    for s in number[n]:
      GPIO.output(seg[s], 1)
    GPIO.output(digit[n-1], 0)
```

The seven-segment display should display the number of the carol. The used function `za(n)` is changed so that the 1 is displayed on the first digit, the 2 on the second, etc. This is not absolutely necessary, but only a graphic effect. To switch off the display completely, a 0 is passed when the function is called.

```
try:
  while True:
    if GPIO.input(t1)==1:
      za(1)
      subprocess.Popen(["omxplayer", "lied1.mp3"])
      time.sleep(4)
      za(0)
```

The main loop of the program asks for all four keys one after the other. If key 1 is pressed, the display shows number 1. Then the song `lied1.mp3` is played. The function `subprocess.Popen()` is used for this, which in this case calls the command line based media player `omxplayer`.

The display switches off after four seconds. The media player runs in its own process. The carol can therefore run even longer. The program doesn't wait. The other carols can be played in the same way.

By renaming the files you can also play other songs. The download contains more than just four Christmas carols.

Merry Christmas!

Vorsichtsmaßnahmen
Auf keinen Fall irgendwelche GPIO-Pins miteinander verbinden und abwarten, was passiert.
Nicht alle GPIO-Pins lassen sich frei programmieren. Einige sind für die Stromversorgung und andere Zwecke fest eingerichtet.
Einige GPIO-Pins sind direkt mit Anschlüssen des Prozessors verbunden, ein Kurzschluss kann den Raspberry Pi komplett zerstören.
Verbindet man über einen Schalter oder eine LED zwei Pins miteinander, muss immer ein Schutzwiderstand dazwischengeschaltet werden.
Für Logiksignale immer Pin 1 verwenden, der +3,3 V liefert und bis 50 mA belastet werden kann. Pin 6 ist die Masseleitung für Logiksignale.
Pin 2 liefert +5 V zur Stromversorgung externer Hardware. Hier kann so viel Strom entnommen werden, wie das USB-Netzteil des Raspberry Pi liefert. Dieser Pin darf aber nicht mit einem GPIO-Eingang verbunden werden.

Warnung! Augenschutz und LEDs:
Blicken Sie nicht aus geringer Entfernung direkt in eine LED, denn ein direkter Blick kann Netzhautschäden verursachen! Dies gilt besonders für helle LEDs im klaren Gehäuse sowie in besonderem Maße für Power-LEDs. Bei weißen, blauen, violetten und ultravioletten LEDs gibt die scheinbare Helligkeit einen falschen Eindruck von der tatsächlichen Gefahr für Ihre Augen. Besondere Vorsicht ist bei der Verwendung von Sammellinsen geboten. Betreiben Sie die LEDs so, wie in der Anleitung vorgesehen, nicht aber mit größeren Strömen.

Liebe Kunden!

CE

Dieses Produkt wurde in Übereinstimmung mit den geltenden europäischen Richtlinien hergestellt und trägt daher das CE-Zeichen. Der bestimmungsgemäße Gebrauch ist in der beiliegenden Anleitung beschrieben.

Bei jeder anderen Nutzung oder Veränderung des Produktes sind allein Sie für die Einhaltung der geltenden Regeln verantwortlich. Bauen Sie die Schaltungen deshalb genau so auf, wie es in der Anleitung beschrieben wird. Das Produkt darf nur zusammen mit dieser Anleitung weitergegeben werden.

Das Symbol der durchkreuzten Mülltonne bedeutet, dass dieses Produkt getrennt vom Hausmüll als Elektroschrott dem Recycling zugeführt werden muss. Wo Sie die nächstgelegene kostenlose Annahmestelle finden, sagt Ihnen Ihre kommunale Verwaltung.

MAKERFACTORY
distributed by Conrad Electronic SE
Klaus-Conrad-Str. 1, 92240 Hirschau
www.makerfactory.com

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.