Raspberry Pi Advent Calendar 2017

Controlling simple electronic systems - even just a few LEDs - with a normal PC or laptop is associated with almost unacceptable effort for the hobby programmer. The PC simply does not have the interfaces required for this. Also, the Windows operating system is pretty much unable to communicate with electronic parts.

The Raspberry Pi is a fully fledged computer - even if it does not look like it at first. Many things are a bit slower than what we are used to from modern PCs but the Raspberry Pi is much smaller and, most importantly, much cheaper than a PC.

This advent calendar contains a hardware experiment with the Raspberry Pi for every day. The experiments are programmed with Scratch or Python. Both programming languages are pre-installed on the Raspberry Pi. All experiments work with Raspberry Pi 3.

The most important components in brief

Patch panel

The advent calendar comes with a patch panel so that electronic circuits can be built quickly without the need for soldering. Electronic components can be attached directly into a breadboard.

All outer longitudinal rows are connected with each other on this patch panel via contacts (X and Y). These contact rows are often used as plus and minus poles to supply the circuitry with power. In the other contact rows, five contacts (A to E and F to J) are each connected, laterally, with a gap in the middle of the board. This allows you to attach larger components and wire them with the surroundings.

The connections on the patch panel.

LEDs

LEDs (short for: light emitting diodes) light up when an electrical current moves through them in the direction specified direction. In circuit diagrammes, LEDs are indicated with a triangle symbol in the shape of an arrow, which shows the direction of the current from the plus pole to the minus pole or the ground wire. One LED conducts almost all of the current in the direction of the specified direction. It only has a very low resistance. To limit the current and, therefore, prevent the LED from blowing, a 220-ohm series resistor is typically installed between the used GPIO pin and the anode of the LED, or between the cathode and the ground pin. This series resistor also protects the GPIO output of the Raspberry Pi from excessive currents. The LEDs in the advent calendar have a series resistor already installed and, therefore, can be connected directly to the GPIO pins.

In what orientation are LED connected?
The two connecting wires of an LED have different lengths. The longer wire is the plus pole or anode, and the shorter one is the cathode. Memory hook: The plus sign has one more line than the minus sign so that the wire for the plus pole should be longer. Furthermore, most LEDs are flattened on the negative side, which is comparable to a minus sign. Another memory hook: cathode = short = edge.

Circuit diagramme of an LED with series resistor.

RGB LEDs

A normal LED always lights up only in one colour. The RGB LEDs in the advent calendar can glow in various colours as selected. In principle, they consist of three differently coloured LEDs installed in a transparent housing. Each of these three LEDs has its own anode, which is connected with one GPIO pin. There is only one cathode that is connected with the ground wire. That is why an RGB LED has four connection wires.

Connection pins of an RGB LED.

The connection wires of RGB LEDs have different lengths so that they can be distinguished clearly. In contrast to normal LEDs, the cathode wire of an RGB LED is the longest.

RGB LEDs work like three individual LEDs and, therefore, require three series resistors. They are also installed in the RGB LEDs contained in the advent calendar.

Resistor

Resistors are used to limit the current in sensitive electronic components and as series resistors for LEDs. The unit for resistors is ohms. 1,000 ohms are one kilo-ohm, short kohm. 1,000 kohms are one mega-ohm, short Mohm. The omega character Ω is often used to denote the unit ohm.

Internal circuit diagramme for an RGB LED with three series resistors.

The coloured rings on the resistors indicate the resistor value. With a bit of practice, they are easier to read than tiny numbers, which can be found on very old resistors.

Most resistors have four of these coloured rings. The first two rings denote numerical digits, the third ring a multiplier and the fourth the tolerance. This tolerance ring is mostly of a gold or silver colour - colours that are not used for the first rings. This makes the reading direction clear. The tolerance value is hardly of importance in digital electronics. The table shows the meaning of the coloured ring on resistors.

The orientation in which a resistor is installed is not important. However, the orientation of LEDs is very important during installation.

Servo

A servo is a small motor with an arm, which can be used to attach objects. It can be rotated by a certain angle with a control signal. Servos have three connection wires. Two of them are for power supply and the servo receives control signals via the third wire.

GPIO connection cable

All coloured connection cables have a connector on both sides, which fits into a GPIO of the Raspberry Pi. LEDs and other components can be connected directly to these connectors. Pin strips are also included, which are attached to the patch board to connect the connection cables with the contact rows. Each pin strip has three pins, which are connected independent of each other, i.e. not electrically.

Allocation of the GPIO pins.

Colour	Resistor value in ohms			
	1. Ring (tens column)	2. Ring (ones column)	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 %
Purple	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	

Precautions
Never simply connect any GPIO pins with each other and wait to see what happens.
Not all GPIO pins can be freely programmed. Some are hard-wired for power supply and other purposes.
Some GPIO pins are directly connected to the connectors of the processor. A short-circuit can completely destroy the Raspberry Pi. However, when connecting two pins with each other via a switch or an LED, a protective resistor must always be installed in between. LEDs with installed series resistor are an exception.
Pin 1 supplies 3.3 V and can be loaded with up to 50 mA. It must always be used for logic signals. Pin 6 is the ground wire for logic signal.
Pins 2 and 4 supply +5 V of power to external hardware. They can provide as much electricity as the USB power adapter of the Raspberry Pi can supply. However, these pins must not be connected with a GPIO input.

Day 1

Today in the advent calendar

- 1 red LED with installed series resistor
- 1 GPIO connection cable

Preparing the Raspberry Pi

To start the raspberry Pi, you will need:

- USB keyboard and mouse
- HDMI cable for the monitor
- Power cable
- MicroSD card with operating system Raspbian Jessie
- Micro USB mobile phone charger as power adapter (at least 2 A)

The power adapter must be the last to be connected. Once connected, the Raspberry Pi will switch on automatically. There is no extra on/off switch.

Installation of the operating system in brief

For all of you who do not have their Raspberry Pi ready for operation with the current Raspbian version, follow this ten-step system installation guide:

1. Download NOOBS (at least version 2.0.0) from **www.raspberrypi.org/downloads** on your PC and unpack the zip archive on the hard drive.
2. Re-format the SD card using the SD formatter on the PC if the SD card has been used before: **www.sdcard.org/downloads/formatter_4**. When doing this, switch on **Format Size Adjustment** (the SD card must have a memory of at least 4 GB).
3. Copy the NOOBS files and directories to the SD card.
4. Remove the SD card from the PC, insert it in the Raspberry Pi and boot. Select **English** as installation language at the very bottom. This will automatically select the English keyboard.
5. Check the box of the pre-selected Raspbian operating system and click **Install** in the top left. After confirming a safety message that the memory card will be overwritten, the installation starts automatically, which will take a few minutes.
6. The Raspberry Pi re-boots when installation is complete.
7. Under **Settings** in the menu, start the **Raspberry Pi Configuration** tool.
8. On the tab **Localisation** in the field **Specify time zone**, select the options **Europe** and **Berlin**. **Language** and **keyboard** should be set to German automatically.
9. On the tab **Interfaces**, set the switch **SSH** to **Activated**, if you want to transfer data from the PC to the Raspberry Pi via the network.
10. Click **OK** and re-boot the Raspberry Pi via the menu item **Shutdown**. A warning may appear regarding an unsafe password, which can be ignored.

Illuminated LED

The experiment shows how LEDs are connected. Pay attention to the correct installation orientation of the LED. The cathode (short wire) is connected to the GND pin of the Raspberry Pi, the anode (long wire) with the GPIO pin 4. Simply plug the LED directly into the GPIO connection cable. All downloads of circuit diagramme drawings are in colour so that you can recognise the wire better.

Components: 1 red LED with installed series resistor, 2 GPIO connection cables

The first LED at the Raspberry Pi is illuminated.

Everything is controlled with a programme in Scratch.

Programme download

The programmes and installation drawings used in the advent calendar can be downloaded here: **www.buch.cd**. Enter the code **15002-8** into the field.

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

Start the file manager on the Raspberry Pi. It shows the home directory automatically when started. Right-click on the downloaded zip-file and select **Unpack here** in the context menu. The archive contains a total of 25 directories. Each day is saved in its own directory. There is also a general directory.

The programme

Scratch is pre-installed on the Raspberry Pi under **Development** in the menu. It is one of the programming languages that is most easy to learn. From Raspbian Jessie onwards, Scratch supports the GPIO interface of the Raspberry Pi. The GPIO support must be activated before first use via the menu item **Edit/Start GPIO server**.

Starting GPIO in Scratch.

You do not need to type code when programming in Scratch. The blocks are simply added to one another with drag-and-drop. The block pallet on the left side of the Scratch window shows the available blocks ordered by theme.

You can create the programme on the screen yourself or you can use programme `01led01` from the download for the advent calendar. To do this, select **Open** in the menu **File** and click on the button **pi** in the next dialogue to select the personal home directory where the downloaded programmes are saved.

This Scratch programme `01led01` makes the LED light up for half a second.

In Scratch, click on the yellow symbol **Control** in the top left. The blocks for control are then shown in the block pallet. For the first programme, we only need these yellow blocks.

Drag the blocks you need simply from the block pallet into the script window in the middle of Scratch.

The block, **if (green flag) clicked,** is used to start a programme. The following script elements are run when clicking on the green flag in the top right in Scratch. The top of the block is round. Thus, it does not fit under another block. It must always be placed first.

The GPIO commands are made with the Scratch block **send... to all**. The respective pin names and key words are entered into the text field. To do this, click into the text field in the block, select **New/edit...** and enter the text.

At first, the GPIO pin 4 is defined as output with **config4out**. Every GPIO pin can be either input or output.

In the next step, LED connected to GPIO pin 4 is switch on with another Scratch block **send... to all** with the text **gpio4on**.

Then the programme waits for half a second. Scratch has its own block **wait...sec** for this. Like many other American programmes, Scratch uses the dot as decimal separator and not the comma as is typical in Germany. Thus, to indicate half a second you need to enter `0.5` and not `0,5`.

In the last step, the LED connected to GPIO pin 4 is switched on with another Scratch block **send... to all** with the text **gpio4off**.

The programme starts when clicking the green flag in the top right of the Scratch window.

Day 2

Today in the advent calendar

- 1 patch board (SYB 46)
- 1 green LED with installed series resistor
- 1 GPIO connection cable
- 2 pin strips (3 pins)

Two LEDs flash alternatingly in red and green

The experiment of Day 2 makes both LEDs light up alternatingly in red and green. Everything is controlled with an infinite loop in Scratch.

Components: 1 patch board, 1 red LED with installed series resistor, 1 green LED with installed series resistor, 3 GPIO connection cables, 2 pin strips (3 pins)

Two LEDs flash at the Raspberry Pi.

This time, and in most other experiments in this advent calendar, we use a patch board to create the circuit. Use the included pin strips to connect the GPIO connection cables to the patch board.

The programme

At first, the two GPIO pins 4 and 10 are defined as outputs with **config4out** and **config10out**.

A **repeat continuously** loop makes the two LEDs flash infinitely until the user clicks on the red stop symbol in the top right in Scratch.

When the red LED at pin 4 is switched on and the green LED at pin 10 is switched off, the programme waits for half a second. Afterwards, the green LED at pin 10 is switched on and the red LED at pin 4 is switched off in the same manner. The cycle is repeated after another half second.

The programme `02led02` controls the two LEDs.

Day 3

Today in the advent calendar

- 2 GPIO connection cable

Two LEDs flash alternatingly in red and green using Python

The experiment of Day 3 again makes both LEDs light up alternatingly in red and green. But this time we use Python instead of Scratch.

Components: 1 patch board, 1 red LED with installed series resistor, 1 green LED with installed series resistor, 3 GPIO connection cables, 2 pin strips (3 pins)

Two LEDs flash at the Raspberry Pi.

Python is pre-installed on the Raspberry Pi, even in two versions. Python 3 is not simply a newer version of Python 3 as the name might suggest. The languages partly use a different syntax. Thus, the programmes are not directly compatible. Many external libraries are available only for one of the two versions. Developers of Python programmes must always tell their users, with which version a programme works. In this advent calendar, we use the modern Python 3.

In the menu under **Development**, start the programme Python 3. IDLE is a complete Python shell and development environment. There are not additional components necessary to start programming.

Via **File/Open** open the programme `03led02.py` from the download or open a new window in the Python shell via **File/New** and type the programme up.

The programme

The programme is almost an exact translation of the Scratch programme on Day 2 but with an important difference: The LEDs do not flash infinitely but exactly ten times.

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

GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.OUT)
GPIO.setup(10, GPIO.OUT)

for i in range(10):
    GPIO.output(4, True)
    GPIO.output(10, False)
    time.sleep(0.5)
    GPIO.output(10, True)
    GPIO.output(4, False)
    time.sleep(0.5)

GPIO.cleanup()
```

This is how the programme works

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

Python programmes that are started via the command line must always start with the line above. This is not necessary for programmes that are only started via the Python shell. Due to compatibility reasons however, you should get in the habit of entering this line at the beginning of each Python programme.

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

A great advantage of python is its extensibility with new functions from function library. For almost every task, there are complete libraries available so that you do not need to work on many standard tasks yourself. The library `EPI.GPIO` is imported to support GPIO pins. The library `time` contains time functions, for example to realise a waiting time between the flashing of the two LEDs.

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

The library `RPI.GPIO` supports two methods to name the pins. In the mode `BCM`, the familiar GPIO port numbers are used that are also used in Scratch In the mode `BOARD` the names correspond to the pin numbers on the Raspberry Pi board. `BCM` is typically used. For this reason and to be able to compare the Python programmes more easily with the Scratch programmes, we will also use the mode `BCM` in this advent calendar.

```
GPIO.setup(4, GPIO.OUT)
GPIO.setup(10, GPIO.OUT)
```

The function `GPIO.setup` initialises a GPIO pin as output or as input. The first parameter names the pin with its GPIO or pin number, according to the `BCM` or `BOARD` mode used. The second parameter can be either `GPIO.OUT` for an output or `GPIO.IN` for an input.

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

Loops with `for` run through a certain number of iterations. In addition to a simple number, a number range of calculation function can also be entered in the parameter `range()`.

Indentations are important in Python.
In most programming languages, programme loops conditionals are indented to make the programme code more structured. In Python, these indentations do not only serve for structural reasons but they are absolutely necessary for the programme logic. In turn, special punctuation characters are not required in this language to end loops or conditionals.

```
    GPIO.output(4, True)
    GPIO.output(10, False)
```

The indented lines are executed once during each loop iteration. The function `GPIO.output` defines that status of a GPIO pin. Each pin can be defined with two different statuses. `True` switches the Pin on, `False` switches it off again. The lines above switch the LED at GPIO pin 4 on and the LED at GPIO pin 10 off.

```
    time.sleep(0.5)
```

The programme waits for half a second.

```
    GPIO.output(10, True)
    GPIO.output(4, False)
    time.sleep(0.5)
```

Then, the LED at GPIO pin 10 is switched on and the LED at GPIO pin 4 switched off. Then the programme waits for half another second.

```
GPIO.cleanup()
```

The last line is not indented. It is only executed after the loop has run ten times. At the end of a programme all used GPIO pins must be reset in order to avoid warnings during the next start of the programme. The line above does this for all GPIO pins initialised by the programme at the same time. Pin initialised by other programmes, however, are not affected. In this way, the execution of other programmes that might run simultaneously is not impaired.

Day 4

Today in the advent calendar

- 2 buttons
- 1 pin strip (3 pins)

Toggling LEDs with a button

The experiment of Day 4 does not toggle the LEDs automatically but only when the user presses a button.

Components: 1 patch board, 1 red LED with installed series resistor, 1 green LED with installed series resistor, 1 button, 5 GPIO connection cables, 3 pin strips (3 pins)

Toggling LEDs with a button.

The programme

The programme works in a similar way to the one on Day 2. Similarly, the two LEDs are switched on and off in an infinite loop. In contrast to the previous programme, the LEDs are not toggled after a certain time but only when the user has pressed the button.

The programme `041ed01` controls the two LEDs via a button.

At first, GPIO pin 7 is defined as input in addition to the two outputs. Inputs at the Raspberry Pi process digital logic signals. If an input with + 3.3 V is connected, it receives the logic signal **High**, which is seen as **1** in Scratch. If the input is connected with GND, it receives the logic signal **Low**, which is seen as **0** in Scratch.

Caution
Never use the +5 V pins of the Raspberry Pi for logic signals in circuits. 5 V would overload the GPIO inputs and damage the Raspberry Pi.

In our circuit, pressing the button connects GPIO pin 7 with +3.3 V. When the button is released, the input is undefined, which must not happen in digital electronics. For such cases, all GPIO pins have so-called pull-down resistors, which define an input without a signal automatically as **Low**.

Define the GPIO pin for the button with **config7inpulldown** in order to activate the inbuilt pull-down resistor at the input.

When an LED is lit, the programme no longer waits for a certain amount of time but uses **wait until...** to wait until a certain event occurs; in this case, until the GPIO pin 7 assumes the value 1, i.e. the button is pressed.

For the query, a long field with pointed ends in the **wait until...** is used. Here, a block from the green block pallet **Operators** must be inserted. Pull the block with the equal sign into the placeholder field in the **wait until...** block.

This operator is always true when both values on the left-hand side and the right-hand side of the equal sign are the same.

In our case, the value of GPIO pin 7 should be 1. The value 1 means **High**. Thus, enter a `1` into the text field on the right in the green block.

Now click the green flag in the top right once to start the incomplete programme. This will define the GPIO pins. Then, click on the red stop sign.

To query GPIO inputs, the block **Value of sensor...** from the blue block pallet **Sensing** is used. Select the sensor **GPIO-7** in the list view of the blue block. In addition to some pre-defined sensors, all GPIO pins are shown that are defined as input. This is why the programme needs to be started briefly once.

Drag the blue block **Value of sensor...** into the left field of the green block inside the block **wait until...**.

The programme will then wait for 0.2 seconds using a block **wait...sec.** This will prevent the button from being recognised as pressed immediately when the programme continues to run. This is the time the use has to release the button.

The LEDs will be toggled only afterwards and the programme waits again until the user presses the button.

Duplicating blocks
When building programmes in Scratch, you do not need to make similar block combination again each time. Right-click on the first block you want to duplicate. The, select **Duplicate** in the menu. All blocks below the selected block will then be duplicated automatically as well. The duplicated blocks can be inserted into the correct place within the programme.

Day 5

Today in the advent calendar

- 2 GPIO connection cable

Toggling LEDs on the Christmas pyramid with Python

The programme of Day 5 uses Python to toggle two LEDs with a button. The LEDs can now be placed on the Christmas pyramid and are connected directly to the Raspberry Pi with GPIO connection cables.

Components: 1 patch board, 1 red LED with installed series resistor, 1 green LED with installed series resistor, 1 button, 6 GPIO connection cables, 1 pin strip (3 pins)

Two LEDs on the pyramid and a button on the patch board.

The Christmas pyramid

On the back of the advent calendar, you can find two parts for cutting out, which make up a Christmas pyramid when assembled.

The two parts of the Christmas pyramid.

Fold the oval part along the printed lines and make small incisions along the short lines on both parts. The round holes in the triangular part are for the LEDs. Assemble the Christmas pyramid as shown in the figure.

Assembling the Christmas pyramid.

The programme

In contrast to yesterday's Scratch programme, the red LED at pin 4 will only be lit as long as the button is pressed in the Python programme `05pyramide01.py`. Once it is released, the green LED at pin 10 will light up instead.

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

GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.OUT)
GPIO.setup(10, GPIO.OUT)
GPIO.setup(7, GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(7)==True:
            GPIO.output(4, True)
            GPIO.output(10, False)
        else:
            GPIO.output(10, True)
            GPIO.output(4, False)

except KeyboardInterrupt:
    GPIO.cleanup()
```

This is how the programme works

```
GPIO.setup(7, GPIO.IN, GPIO.PUD_DOWN)
```

After the already familiar initialisation of the GPIO pins for the LEDs, pin 7 is initialised as input for the button. The parameter `GPIO.PUD_DOWN` switches the internal pull-down resistor on.

```
try:
    while True:
```

These lines start an infinite loop with the option to end it. The `while` loop runs infinitely because the condition is always `True`. However, the earlier `try` command allows a break condition specified in `except` further down.

```
        if GPIO.input(7)==True:
            GPIO.output(4, True)
            GPIO.output(10, False)
        else:
            GPIO.output(10, True)
            GPIO.output(4, False)
```

The word `if` indicates a conditional. If the condition is met, the following indented part of the programme will be executed.

== is not the same as =
A double equal sign `==` is a comparison operator while a single equal sign `=` is used to assign variables.

When the button at pin 7 is pressed, it has the value `True`. The LED at pin 4 is switched on, the LED at pin 10 is switched off.

Another block with the key word `else` can be added after the programme part that will executed if the condition is met. The following part of the programme will be executed if the condition is not met.

If the button at pin 7 is not pressed, the LED at pin 10 is switched on and the LED at pin 4 is switched off.

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

Pressing the shortcut [Ctrl]+[C] on the keyboard initiates the break condition. This will end the loop and the GPIO pins will be reset subsequently.

Day 6

Today in the advent calendar

· 1 yellow LED with installed series resistor

The Scratch cat controls the LEDs

The cat on the Scratch stage (in the top right window) is not only the Scratch logo but a freely programmable object that can execute different tasks. In the experiment of day 6, the cat will walk past a background with three differently coloured strips - green, yellow and red. Depending on the background colour the cat touches, the corresponding LED will light up.

Components: 1 patch board, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 4 GPIO connection cables, 2 pin strip (3 pins)

The Scratch cat toggles three LEDs

The programme

The programme uses the so-called Scratch stage, a graphical task window in the top right of the Scratch screen. The background image can be painted with the paint programme that is included in Scratch.

The programme `06cat01` allows the Scratch cat to control three LEDs.

Paint programme for the background in Scratch.

To change the background, click on the white symbol **Stage** in the object pallet in the bottom right of the Scratch screen.

The Scratch programme window is empty because programme blocks cannot be used directly for the stage. Click on the tab **Backgrounds** above and then on **Edit**. This will start the paint programme, which provides the basic functions of a simple graphic programme.

Paint three coloured bars in green, yellow and red onto the background using the rectangular symbol and the colour pallet in the bottom left.

When you are done, exit the paint programme with **OK**, click on the cat in the object pallet in the bottom right (here named **Object1**) and go back to the tab **Scripts** where you can build the programme.

After the start, the three GPIO pins for the LEDs are first defined as outputs: **config4out**, **config22out**, **config11out**.

The, the cat is placed in its start position at the left side of the stage on the green bar. To do this, use block **go to x:... y:...** from the block pallet **Movement**. This will place the cat to an absolute coordinate position.

The horizontal edge of the Scratch stage is the x-direction and its coordinates range from -240 to 240. By default, the cat is placed at 0,0, which is the exact centre. The cat itself has a width of about 100 coordinate units so that the coordinates **x:-150 y:0** entered into the block will give it a good starting position to walk along the stage.

After that, a **Repeat continuously** loop will start, which makes the cat move infinitely.

Another loop runs within this loop, which is repeated exactly 30 times. Since the cat is to move ten coordinate units in x-direction, it will reach the turning point **x:150 y:0** at the right-hand side of the stage after 30 iterations.

Three **if...else** conditionals are placed inside this loop. Each of them checks if the cat touches a certain background colour. If this is the case, the block inside the top brackets **if** is executed. If not, the other block inside the bottom brackets **else** is executed.

To check the colour, we use the block **if colour... is touched** from the block pallet **Sensing**. Clicking on the coloured image in the block will turn the mouse pointer into a pipette and you can select the desired colour on the Scratch stage by clicking on it.

The first **if...else** conditional checks, if the cat touches the green colour of the bar in the middle. If this is the case, the green LED at pin 11 is to be switched on (**gpio11on**). If the cat does not touch the green bar at the time the conditional is run, this LED is switched off (**gpio11off**).

In the same way, the second **if...else** conditional switches the yellow LED at pin 22 on and off when the yellow bar is touched. The third

if...else conditional switches the red LED at pin 4 on and off when the red bar is touched.

Now the cat will move ten coordinate units forwards. The block **go...steps** from the block pallet **Movement** moves an object over a relative distance starting from its current position. This is different to the block **go to x:... y:...** used at the start, which moves the object to an absolute coordinate position.

In order to be able to follow the movement, the programme waits for 0.2 seconds after each step before starting the loop from the beginning and the cat takes the next step.

When the cat reaches the right-hand side of the stage after 30 steps, it should turn 180 degrees. To do this, use block **turn... degrees** from the block pallet **Movement**. Activate the double-arrow symbol near the cat in the top area of the script window so that the cat will not turn its head.

Subsequently, the infinite loop will start and the cat will move back. Since only relative movements are used, it is not necessary to distinguish between moving to the right or moving to the left in the script.

This symbol toggles turning and left-right orientation.

Day 7

Today in the advent calendar

· 2 GPIO connection cable

Controlling LEDs with a smartphone app

Some circuits you build with this advent calendar are controlled by a mobile app. For this purpose, a mobile webpage is programmed and accessed with the smartphone. The mobile webpage works on every smartphone using a Wi-Fi connection and the installed browser but you can also use the programmed page with the browser on the Raspberry Pi or on your PC. It is important that smartphone and Raspberry Pi are connected to the same network. In addition, the following must be done:

· **Step 1:** Install the web server

· **Step 2:** Install PHP5

Step 1: Install the web server

You must install a web server so that the webpage can run on the Raspberry Pi and accessed with your smartphone. We use the Apache HTTP server. Open the **LXTerminal** from the menu **Accessories** for the installation.

Enter the following command in the open LXTerminal window:

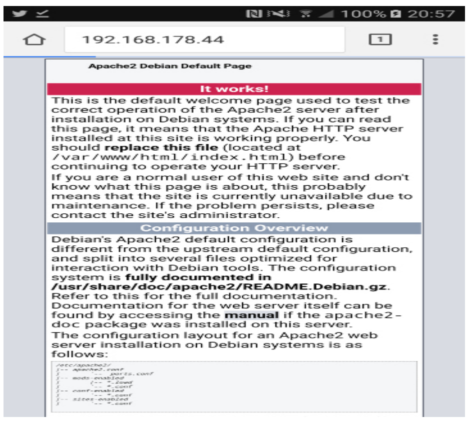
```
sudo apt-get install apache2
```

The web server will now be downloaded and installed. You can test the installation directly via your smartphone (or via the browser on the Raspberry Pi). To do this, you will need the IP address of the Raspberry Pi. You can find it when hovering the mouse over the network symbol in the top right of the desktop.

Open the browser in your smartphone and enter `http://<IP-Adresse Raspberry PI>`. The example from the screenshot shows the address **http://192.168.178.44**.

The prompt window opens when clicking on the menu entry.

The Raspberry Pi desktop will display the IP address.


192.168.178.44

Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `-- ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

The test page on the smartphone. The web server installation is now complete.

The address on your Raspberry Pi will most likely be different. You can also test the web server using the browser on your Raspberry Pi. To do this, enter the address `http://localhost` and the test page will be shown in the browser.

Step 2: Install PHP5

The command `phpinfo()` provides system information for PHP installation in form of a webpage.

After the web server was tested successfully, you now need to install PHP5. First, start the LXTerminal and update the system with the following command:

```
sudo apt-get update
```

Install PHP5 with the following command:

```
sudo apt-get install php5
```

Test the installation afterwards. To do this, stay in the LXTerminal. Go to the directory with the command `cd/var/www` and add writing rights for all users for the HTML directory `/var/www/html`:

```
chmod -R a+w html
```

Go to the HTML directory with `cd html` and create and use `touch phpinfo.php` to create an empty file with the name `phpinfo.php`. Open this file for editing with the command `nano phpinfo.php`. Then type the following source code into the file:

```
<?php
  phpinfo();
?>
```

Save the file with the keyboard shortcut [Ctrl]+[O]. The editor will ask you for a name. Confirm the suggestions with the [Enter] button. Close the editor with [Ctrl]+[X].

Open the file in the browser of your smartphone via **http://<IP-Adresse-Raspberry-Pi>/phpinfo.php** in the browser of the Raspberry Pi via **http://localhost/phpinfo.php**.

Building the project

We now have all the requirements to realise the first mobile app. First, we need to assemble the circuit. To do this, we need the following components.

Components: 1 patch board, 1 red LED with installed series resistor, 2 GPIO connection cables, 2 pin strips (3 pins)

An LED is switched on with the app and it switches off automatically after a pre-defined time.

The programme

The Python programme `07led.py` from the directory `day07` is used to control the LED. The LED switches on and switches off after one second. You can change the lighting duration in the function `time.sleep(1)`. The parameter is the duration in seconds, i.e. one second in this case.

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

GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.OUT)
GPIO.output(4, True)
time.sleep(1)
GPIO.cleanup()
```

The app

You can create the mobile webpage based on a template. You can find the template in the directory `app_template` in the download archive. The directory contains the file `index.php` and the directory `css`. The directory `css` contains the format templates for the mobile app. The file `index.php` is the actual template for the app. You will need this template on various other days.

Step 1: Copying the template into the web server directory

The content of `/var/www/html/day07` after copying the template.

Open the LXTerminal and go to the directory `/var/www/html`:

```
cd /var/www/html
```

In this directory, create a new directory with the name `day07`:

```
mkdir day07
```

Go to this directory and copy the content of the directory `app_template` and the file `07led.py` into the directory you just created. The directory should now look as shown in the adjacent figure.

You can now open the template in the browser on the Raspberry Pi via **http://localhost/day07/index.php**.

Step 2: Customising the template

The copied template in the browser on the Raspberry Pi.

Then open the file `index.php` from the directory `day07` by entering `nano index.php` into the LXTerminal. In the file, you can see, amongst others, the following block:

```
 <form action="index.php" method="post">
</form>
```

This block is a form with which you can send commands to the web server and, therefore, to the Raspberry Pi. To switch the LED on, you need a button. Thus, change the block as follows:

```
<form action="index.php" method="post">
<input name="light_on" id="light_on" type="submit" value="Light on" style="background-color: #4180C5">
</form>
```

This will create a blue button with the text `Light on`. If you want to change the colour, change the value in `background-color`. To change the text, adjust the text in `value`. Save the file with [Ctrl]+[O] and close the editor with [Ctrl]+[X]. Using **http://localhost/day07/index.php** , open the file `index.php` again in the browser.

The customised template in the browser on the Raspberry Pi.

The template now only needs to be linked to the Python script. The script is opened with PHP. Change the beginning of the file `index.php` as follows:

```
<head>
  <?php
   if ($_POST["light_on"]) {
  echo shell_exec('sudo /var/www/html/day07/07led.py');
}
  ?>
  <title>Conrad Raspberry Pi Advent Calendar</title>
```

Open the file again with `nano index.php`. The PHP part is initiated with `<?php` and ended with `?>`. When the button **Light on** is pressed, a post request is sent, which is processed by the page. You can view the sent values with `$_POST`. For the definition of the button, the ID and the name of the button are set to `light_on`:

```
<input name="light_on" id="light_on" type="submit" value="Light on" style="background-color: #4180C5">
```

The `if` conditional checks with `$_POST["light_on"]` if the button was pressed (the value is only sent once in this case). In this case, the Python script is requested with `echo shell_exec`. And now the mobile webpage is complete.

When the page is opened in the browser and the button is pressed, nothing happens. You need to define the correct authorisations.

Step 3: Defining authorisations

Open the LXTerminal again and go to the directory of Day 7 with `cd /var/www/html/day07`. Provide the file with the execution authorisation:

```
chmod a+x 07led.py
```

Now you need to authorise the web server to run this script. Use the following command for this:

```
sudo visudo
```

Then add this line:

```
www-data ALL=(ALL) NOPASSWD: /var/www/html/day07/07led.py
```

Save the file with [Ctrl]+[O] and close it with [Ctrl]+[X].

Step 4: Testing the app

You did it! Now you can test the app on your smartphone. Start the browser in your smartphone, open the page with **http://<IP-Adresse Raspberry Pi/day07/index.php** and press the button **Light on**. The light should go on.

You can now switch on the light with your smartphone.

Day 8

Today in the advent calendar

· 1 button

Dimming the LEDs

LEDs can be in one of two conditions: on and off. The same is true for the GPIO ports defined as digital outputs. Thus, it would theoretically not be possible to dim an LED.

However, there is a trick with which the brightness of an LED connected to a digital GPIO port can be controlled. If you make an LED flash very fast, the human eye cannot perceive the flashing. The method is called pulse with modulation and generates a pulsing signal, which switches on and off in very short intervals. The signal voltage remains constant, only the ratio between the **False** level (0 V) and the **True** level (+3.3 V) is changed. The key ratio defines the ratio of the length of the switched-on condition and the total duration of the switching cycle. The smaller the ratio, the shorter is the duration when the LED is lit during a switching cycle. This makes the LED appear darker than a permanently lit LED.

Components: 1 patch board, 1 yellow LED with installed series resistor,
button, 2 GPIO connection cables, 3 pin strips (3 pins)

Left: key ratio 50 % - right: key ratio 20 %.

The two buttons dim the LED.

One of the GPIO connection cables is used to connect two contact rows of the patch board with which the two buttons are connected to the +3.3 V-Pn of the Raspberry Pi.

The programme

One of the two buttons makes the LED light up more brightly in steps, the other buttons makes it darker every time it is pressed.

Variables in Scratch
Variables are small memory spaces, in which a programme stores a number or something else. When the programme is closed, these variable memory spaces are erased automatically. In Scratch, variables first need to be defined in the block pallet **Variables** before they can be used. Subsequently, you can drag the symbol of the new variable from the block pallet into a designated field of a block in the programme. There are also various blocks for reading and adjusting of variables on the block pallet.

The programme `08pwm01` dims an LED with two buttons.

First, the programme defines the GPIO pin 4 as output with PWM function. For a pin defined in this way, Scratch offers special function to control a PWM signal.

Then the previously defined variable **pwm** is assigned the value **0**. In this state, the PWM pin is completely switched off. After that, an infinite loop queries the two buttons, changes the variables **pwm** if one of them was pressed and subsequently assigns the GPIO pin 4 a new value.

The **if** conditionals use an **and** block to check two conditions that must be met to increase or decrease the value of the variable **pwm**.

On the one hand, the button must be pressed. On the other, the variable **pwm** must not be outside a threshold range between 500 and 0. For this reason, it is checked whether the value is still less than 500 when increasing and whether it is greater than 0 when decreasing.

For each variable, a block is created on the block pallet **variables**. You can use this block in other Scratch blocks to evaluate the content of this variable.

The block **change... by...** changes a variable relative to its current value. In our case, 10 is added or subtracted.

Regardless of the event resulting from the two **if** conditionals, the GPIO pin then reset to the current value of the variable.

To do this, we use the block **connect ... ...**, which connects any two texts by placing one behind the other. Here, numbers are treated like text, i.e. they are not added but also listed one after another.

Type the characters `gpio4pwm` into the first field of the block and drag the variable **pwm** into the second field. This will assign a certain value to a PWM pin.

The LED lights up with the pre-set brightness. The infinite loop then runs through another iteration and queries if a button was pressed.

Day 8

Today in the advent calendar

· 1 pin strip (3 pins)

Dimming LEDs with Python

The experiment of Day 9 dims an LED with two buttons using a Python programme. The principle is similar to the Scratch programme on Day 8.

Components: 1 patch board, 1 yellow LED with installed series resistor, 2 buttons, 6 GPIO connection cables, 3 pin strips (3 pins)

The two buttons dim the LED.

The programme

The programme `09pwm01py` shows how PWM signals can be output with Python.

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

GPIO.setmode(GPIO.BCM)
LED = 4
T1 = 16
T2 = 7

GPIO.setup(LED, GPIO.OUT)
GPIO.setup(T1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(T2, GPIO.IN, GPIO.PUD_DOWN)

pwm = 0
p = GPIO.PWM(LED, 50)
p.start(pwm)

try:
  while True:
    if GPIO.input(T1)==True and pwm<100:
      pwm += 10
    if GPIO.input(T2)==True and pwm>0:
      pwm -= 10
    p.ChangeDutyCycle(pwm)
    time.sleep(0.1)

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

This is how the programme works

```
LED = 4
T1 = 16
T2 = 7
```

This time, we will use variables for the numbers used for the GPIO pins of the LED and the two buttons. This makes the programme more structured and it shows immediately clear which pin is used. Furthermore, the circuit can be modified easily. All you have to do is to change the respective pin in one place.

```
p = GPIO.PWM(LED, 50)
```

The function `GPIO.PWM()` from the GPIO library is crucial for the outputting of PWM signals. This function requires two parameters - the GPIO pin and the frequency of the PWM signal. In our case, the GPIO pin is defined with the variable `LED`, the frequency is 50 Hertz (oscillations per second).

Why 50 Hertz is the ideal frequency for PWM
The human eye cannot perceive light changes at frequencies higher than 20 Hertz. Since the DC grid in Europe uses a frequency of 50 Hertz, many bulbs flash with this frequency. An LED flashing with more than 20 Hertz but less than 50 Hertz may cause interferences with other light sources so that the dimming effect no longer appears to be even.

`GPIO.PWM()` creates a so-called object, which is stored in the variable `p`. Such objects are more than just simple variables. Objects can have different properties and can be influenced with methods. Methods are specified behind the object name separated by a dot.

```
p.start(pwm)
```

The method `start()` starts the generation of the PWM signal. To do this, you need to specify a key ratio. In our case, the key ratio is `0`, which was stored before in the variable `pwm`. Thus, the LED is always switched off. In contrast to Scratch where PWM values are between 0 and 500, Python uses values between 0 and 100, which refer to the percentage of the duration during which the pin is switch on during a frequency cycle. Hence, a PWM value of 25 makes the LED light up for a quarter of the time and it is switched off during the remainder of the cycle time.

```
try:
  while True:
    if GPIO.input(T1)==True and pwm<100:
      pwm += 10
    if GPIO.input(T2)==True and pwm>0:
      pwm -= 10
```

Then, an infinite loop starts again, which first queries the two buttons. Similar to the Scratch programme, it changes the variable `pwm` as long as the threshold values are not exceeded.

```
    p.ChangeDutyCycle(pwm)
```

In every loop iteration, the method `ChangeDutyCycle()` changes the key ratio of the PWM object to the current value of the variable `pwm`.

```
    time.sleep(0.1)
```

Afterwards, the programme waits for 0.1 seconds until the next loop iteration. This ensures that accidentally pressing the button for longer is not interpreted as pressing the button several times.

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

Pressing the shortcut [Ctrl]+[C] on the keyboard terminates the PWM signal before resetting the GPIO pins.

Day 10

Today in the advent calendar

- 2 GPIO connection cable

Dimming LEDs with an app

The experiment of Day 10 dims an LED with a mobile app. You need the following components to build the circuit:

Components: 1 patch board, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 4 GPIO connection cables

The two LEDs are dimmed with a mobile app.

The programme

The LEDs are dimmed with an app. There will be one button for increasing (up) and decreasing (down) the brightness. A Python script will run when the button is pressed. Since there are two actions, we need two programmes. For increasing the brightness, use the programme `10pwm_up.py`:

```
#!/usr/bin/python
#10pwm_up.py
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
LED = 4
GPIO.setup(LED, GPIO.OUT)

pwm = 0
p = GPIO.PWM(LED, 50)
p.start(pwm)

while pwm<100:
  pwm += 5
  p.ChangeDutyCycle(pwm)
  time.sleep(0.1)

p.stop()
GPIO.cleanup()
```

To decrease the brightness, use the programme `10pwm_down.py`:

```
#!/usr/bin/python
#10pwm_down.py
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
LED = 22
GPIO.setup(LED, GPIO.OUT)

pwm = 100
p = GPIO.PWM(LED, 50)
p.start(pwm)

while pwm>0:
  pwm -= 5
  p.ChangeDutyCycle(pwm)
  time.sleep(0.1)

p.stop()
GPIO.cleanup()
```

The app

If you have not installed the web server and PHP5, please follow the steps outline in Day 7.

To implement the app, you need to go through the same four steps as on Day 7.

Step 1: Copying the template into the web server directory

First, you need to copy the template `app_template` into a directory on the web server. To do this, follow Step 1 for the app on Day 7. This time, name your directory `day10` instead of `day07`.

Step 2: Customising the template

Open the file `index.php` from the directory `day10` by entering `nano index.php` into the LXTerminal. Add the required buttons to the empty block `form`:

```
<form action="index.php" method="post">
 <input name="up" id="up" type="submit" value="Increase brightness"
   style="background-color: #4180C5">
 <input name="down" id="down" type="submit" value="Decrease brightness"
   style="background-color: #4180C5">
</form>
```

This will create two blue buttons with the text `Increase brightness` and `Decrease brightness`. Save the file with [Ctrl]+[O] and close the editor with [Ctrl]+[X]. Then, open the file `index.php` again in the browser using **http://localhost/day10/index.php**.

The template now only needs to be linked to the two Python scripts. To do this, change the beginning of the file `index.php` as follows:

```
<head>
 <?php
  if ($_POST["up"]) {
   echo shell_exec(‘sudo /var/www/html/day10/10pwm_up.py’);
  } elseif ($_POST["down"]) {
   echo shell_exec(‘sudo /var/www/html/day10/10pwm_down.py’);
  }
 ?>
<title>Conrad Raspberry Pi Advent Calendar</title>
```

The mobile webpage is not complete and you can assign the authorisations.

Step 3: Defining authorisations

Open the LXTerminal again and go to the directory of Day 10 with `cd /var/www/html/day10`. Provide the scripts with the execution authorisation:

```
chmod a+x 10*.py
```

Now you need to authorise the web server to run this script. Use the following command for this:

```
sudo visudo
```

Insert the following lines:

```
www-data ALL=(ALL) NOPASSWD: /var/www/html/day10/10pwm_up.py
www-data ALL=(ALL) NOPASSWD: /var/www/html/day10/10pwm_down.py
```

Save the file with [Ctrl]+[O] and close it with [Ctrl]+[X].

Step 4: Testing the app

You did it! Now you can test the app on your smartphone. Start the browser in your smartphone, open the page with **http://<IP-Adresse Raspberry Pi/day10/index.php** and now you can change the brightness of the light.

You can now dim the light with your smartphone.

Day 11

Today in the advent calendar

· 1 servo

Controlling the servo

A servo is a small motor, which rotates a servo arm or other moveable parts into a certain direction. Most servos have a rotatory range of 180 degrees, i.e. they do not execute several subsequent rotations as traditional motors do.

Components: 1 servo, 1 pin strip (3 pins), 3 GPIO connection cables

Controlling the servo with a Scratch programme.

In this experiment, the servo is connected without a patch board directly to the Raspberry Pi using the GPIO connection cables and a pin strip. The red and the black wires are used for power supply, the yellow wire transmits the control signals to the servo.

The programme

Scratch has specific codes for controlling servos. In other programming languages, this is achieved with PWM signals.

The programme `11servo01` controls a servo via keyboard inputs.

This time, the programme consists of several independent blocks, which are activated with different actions. Clicking on the green flag in Scratch defines pin 18 as output for the control signal of the servo and the variable **s** is set to 0. It is used for the control signal.

Scratch sends a control signal to the servo, which is a percentage of the distance to the end stop in one direction. 0% indicates the centre position of the servo, 100% is at one end stop (usually around 90 degrees) and -100% at the other end stop.

Via a block **connect... ...** the signal **servo18%0** is send directly after initialisation. Here, 18 refers to the GPIO pin and 0 to the servo angle.

Three blocks **If key ... is pressed** start automatic actions, if the corresponding key was pressed on the keyboard. Programme loops that wait for a specific action are not needed in Scratch; they are only needed for connected buttons.

The key [Arrowæ] assigns the variable **s** the value **-100**, which rotates the servo to one end stop. The key [ArrowÆ] assigns the variable s the value **100**, which rotates the servo to the other end stop. The key [Arrow½] rotates the servo to the centre position.

Servos tend to not stay still in a certain position but vibrate slightly. The [Space bar] sends the command **servo18stop** and then switches the GPIO pin off. The servo stops. The programme then needs to be activated again with the green flag in order to be able to control the servo.

Day 12

Today in the advent calendar

- 1 servo arm with mounting screws

Screw the servo arm to the top of the servo using the short screw.

Rotating the servo with Python

The experiment of Day 12 controls the servo with two buttons. Pressing one button rotates the servo in one step to the left; pressing the other buttons rotates the servo in one step to the right.

Components: 1 servo, 2 buttons, 4 pin strip (3 pins), 6 GPIO connection cables

The servo is controlled with two buttons.

When building the circuit, pay careful attention to the rows on the patch board shown in the figure. The +3.3 V signal of the servo is used as High signal for the buttons at the same time.

The programme

The programme `12servo03.py` is similar to the programme for dimming of an LED with two buttons. Since Python does not offer its own functions for servos, PWM signals must be used similar to controlling LEDs.

For a 50 Hertz PWM signal, a value of 2.5 corresponds to one end of the rotatory range of the servo and 12.5 corresponds to the other end.

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

servoPIN = 18
t1 = 16
t2 = 7
pwm = 7.5

GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)
GPIO.setup(t1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

p = GPIO.PWM(servoPIN, 50)
p.start(pwm)

try:
  while True:
    if GPIO.input(t1) == True and pwm < 12.5:
      pwm += 2.5
      p.ChangeDutyCycle(pwm)
      time.sleep(0.5)
      p.ChangeDutyCycle(0)
    if GPIO.input(t2) == True and pwm > 2.5:
      pwm -= 2.5
      p.ChangeDutyCycle(pwm)
      time.sleep(0.5)
      p.ChangeDutyCycle(0)

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

This is how the programme works

```
servoPIN = 18
t1 = 16
t2 = 7
pwm = 7.5
```

The variables for the GPIO pins used are defined at the start of the programme. The variable `pwm` is set to a PWM value of 7.5, which corresponds to the centre position of the servo.

Initialising the GPIO pins and the PWM signal is done in the same way as for dimming of LEDs.

```
    if GPIO.input(t1) == True and pwm < 12.5:
      pwm += 2.5
      p.ChangeDutyCycle(pwm)
      time.sleep(0.5)
      p.ChangeDutyCycle(0)
```

The buttons are checked again inside the infinite loop. If a button was pressed and the limit value has not been reached yet, the variable `pwm` is adjusted by 2.5 units in the corresponding direction.

Afterwards, the PWM signal is set to the new value by `p.ChangeDutyCycle(pwm)` and a new control impulse is sent to the servo.

The PWM signal is set to 0 after a waiting time of half a second. This prevents vibrating of the servo. This can happen because a software-generated PWM signal always has very small tolerances.

Pressing the shortcut [Ctrl]+[C] on the keyboard stops the PWM signal and the PWM pins are reset.

Day 13

Today in the advent calendar

- 1 portion modelling clay

Shape the modelling clay into a foot for the servo so that it does not topple over. The modelling clay is also used for experiments later on.

Controlling the servo with an app

Today, the circuit built on day 11 is used. The servo is now controlled by a mobile app.

Components: 1 servo, 1 pin strip (3 pins), 3 GPIO connection cables

You already built the circuit the day before yesterday.

The programme

The servo is rotated with an app. There will be one button rotating to the right and left. The corresponding Python script will run when the button is pressed. Since there are two actions, we need two programmes. For rotations to the left, use the programme `13servo_le.py`:

```
#!/usr/bin/python
#13servo_le.py
import RPi.GPIO as GPIO
import time

servoPIN = 18
pwm = 12.5

GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)
p = GPIO.PWM(servoPIN, 50)
p.start(pwm)
time.sleep(0.5)
p.ChangeDutyCycle(0)
p.stop()
```

```
GPIO.cleanup()
```

For rotations to the right, use the programme `13servo_ri.py`:

```
#!/usr/bin/python
#13servo_re.py
import RPi.GPIO as GPIO
import time

servoPIN = 18
pwm = 2.5

GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)
p = GPIO.PWM(servoPIN, 50)
p.start(pwm)
time.sleep(0.5)
p.ChangeDutyCycle(0)
p.stop()
GPIO.cleanup()
```

The app

If you have not installed the web server and PHP5, please follow the steps outline in Day 7.

To implement the app, you need to go through the same four steps as for the app on Day 7.

Step 1: Copying the template into the web server directory

First, you need to copy the template `app_template` into a directory on the web server. To do this, follow Step 1 for the app on Day 7. This time, name your directory `day13` instead of `day07`.

Step 2: Customising the template

Open the file `index.php` from the directory `day13` by entering `nano index.php` into the LXTerminal. Add the required buttons to the empty block `form`:

```
<form action="index.php" method="post">
<input name="servo_left" id="servo_left" type="submit"
  value="Servo left" style="background-color: #4180C5">
<input name="servo_right" id="servo_right" type="submit"
  value="Servo right" style="background-color: #4180C5">
</form>
```

This will create two blue buttons with the text `Servo left` and `Servo right`. Save the file with [Ctrl]+[O] and close the editor with [Ctrl]+[X]. Then, open the file `index.php` again in the browser using **http://localhost/day13/index.php**.

The template now only needs to be linked to the two Python scripts. To do this, change the beginning of the file `index.php` as follows:

```
<head>
 <?php
  if ($_POST["servo_left"]) {
   echo shell_exec('sudo /var/www/html/day13/13servo_le.py');
  } elseif ($_POST["servo_right"]) {
   echo shell_exec('sudo /var/www/html/day13/13servo_ri.py');
  }
?>
<title>Conrad Raspberry Pi Advent Calendar</title>
```

The mobile webpage is now complete and you can assign the authorisations.

Step 3: Defining authorisations

Open the LXTerminal again and go to the directory of Day 13 with `cd /var/www/html/day13`. Provide the scripts with the execution authorisation:

```
chmod a+x 13*.py
```

Now you need to authorise the web server to run this script. Use the following command for this:

```
sudo visudo
```

Insert the following lines:

```
www-data ALL=(ALL) NOPASSWD: /var/www/html/day13/13servo_le.py
www-data ALL=(ALL) NOPASSWD: /var/www/html/day13/13servo_re.py
```

Save the file with [Ctrl]+[O] and close it with [Ctrl]+[X].

Step 4: Testing the app

You did it! Now you can test the app on your smartphone. Start the browser in your smartphone, open the page with **http://<IP-Adresse Raspberry Pi/day13/index.php** and now you can rotate the servo.

You can now control the servo with your smartphone.

Day 14

Today in the advent calendar

- 1 20-Mohm resistor (red-black-blue)
- 1 pin strip (3 pins)

Sensor contact made with modelling clay

Traffic lights, door openers, light switches and automats these days are often controlled with sensor contacts, which only need to be touched. Buttons that actually need to be pressed are becoming rarer. The experiment of Day 14 controls a game of chance with a simple sensor contact.

Components: 1 patch board, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 1 20-Mohm resistor (red-black-blue), 7 GPIO connection cables, 5 pin strips
(3 pins), one modelling clay contact

This is how sensor contacts work

The GPIO pin defined as input is connected to +3.3 V via a resistor with very high impedance (20 Mohms) so that a weak but clearly as High defined signal is present. Unless a person levitates above the ground, they are always grounded and provide a Low signal via the electrically conducting skin. If they touch a sensor contact, the weak High signal is superimposed by the clearly stronger Low signal of the Hand and makes provides a Low signal at the GPIO pin.

The actual magnitude of impedance between the hand and the ground is dependent on many things such as shoes or the type of floor. The connection to the ground is best when as person stands barefoot on grass but stone floor also mostly works well. Wooden floors are stronger insulators and synthetic carpets are often even charged positively. If the sensor contact does not work, touch the pin strip on the ground strip of the patch board and the actual sensor at the same time. This will definitely create a ground connection.

Modelling clay conducts electricity approximately, as well as human skin. It can be shaped into any form and a modelling clay contact is more pleasant to touch than a simple piece of wire. The area between the hand and the contact is considerably larger. This prevents loose contacts fairly well. Insert a pin strip in to a bit of modelling clay and connect a GPIO connection cable to it. Connect its other end to the patch board using another pin strip. If you need the ground contact, you can also press a piece of modelling clay onto the pins pointing upwards to create a better contact to the hand.

The sensor contact switches the LED on.

The programme `14clay01` makes three LEDs flash in short succession.

The programme

The programme `14clay01` makes three LEDs flash in short succession as long as the clay contact is touched. Once the contact is interrupted, a random LED will light up.

For the sensor contacts to work, the internal pull-up resistor at GPIO pin 20, which is always switched on at the Raspberry Pi by default must be switched off first. This is done with the addition of **pullnone** at the GPIO command **config20in** to initialise the GPIO input at the start of the programme.

In the main loop of the programme, a block **wait until...** waits for the value of the GPIO pin 20 become 0 due to the modelling clay contact being touched.

After that, a **repeat until...** loop is executed until the value of the GPIO pin 20 changes to 1 doe to the contact to the clay being interrupted. Inside the loop, the three LEDs are switched on and off briefly in succession.

When removing the hand from the clay contact, all LEDs are always switched off because the contact is checked only at the start of the loop iteration. In this case, we want the loop to end and a randomly selected LED to light up.

GPIO pins with successive numbers are used in circuit for the used randomisation algorithm. In this way a number between 9 and 11 can be selected randomly.

The programme generates a random number between 9 and 11. Two nested **connect** blocks generate one of these character sequences: **gpio9on**, **gpio10on**, **gpio11on**.

The LED switched on by the character sequence remains switched on when the infinite loop starts running again and waits for the clay contact to be touched. The flashing cycle switches off each LED once after a complete cycle of 0.05 seconds at the latest. Therefore, a specific instruction to switch off the randomly activated LED is not part of the programme.

How are random numbers generated?
The general opinion is that nothing can happen randomly in a programme. So how can a programme be able to generate random numbers? When a large prime number is divided by an arbitrary number, the nth decimal place of the resulting number will be almost impossible to predict. They also change without much regularity when the divisor is increase frequently. This result seems random but can always be reproduced by an identical programme or by executing the same programme several times. However, if you take a number created by these digits and divide it again by another number corresponding to the current seconds of the time or the content of an arbitrary memory space of the computer, you will get a result that cannot be reproduced and, therefore, is called a random number.

Day 15

Today in the advent calendar

- 1 RGB LED with installed series resistor

Play of RGB colours

The experiment of Day 15 makes an RGB LED light up automatically in different colours in succession.

Components: 1 patch board, 1 RGB LED with installed series resistors, 4 GPIO connection cables, 3 pin strips (3 pins)

RGB LED at the GPIO pins 23, 24, 25.

The programme

The programme `15rgbled01` switches different colours of the three colour components of the RGB LED on and off in succession. Since two colours are switched on for a duration, the RGB LED lights up alternatingly in different mixed colours.

The programme `15rgbled01` controls an RGB LED.

First, the variable **z** is set to 5. It controls the time between lighting of the individual colours and, therefore, the flashing frequency of the RGB LED. The variable is shown with a controller on the Scratch stage. The user can change the frequency interactively while the programme is running. These controllers can be shown and hidden with a right-click on the variable on the Scratch stage. Since the controller can only be set to whole numbers, the variable **z** shows the time in tenths of a second.

After initialising the used GPIO outputs, pin 24 is the only one that is switched on. The RGB LED lights up in green. After that, other pins are switched on and off on after another. The waiting time between the switches is controlled by the variable **z** in tenths of a second and can be changed interactively by the user with the controller on the Scratch stage.

Day 16

Today in the advent calendar

- 1 RGB LED with installed series resistor

RGB colour mix

The experiment of Day 16 shows different colour changes on two RGB LEDs. The colour changes can be altered interactively by the user on the monitor. Mixed colours and colour changes occur when the individual colours of an RGB LED are given different brightnesses with PWM signals.

Components: 1 patch board, 2 RGB LEDs with installed series resistors, 7 GPIO connection cables, 5 pin strips (3 pins)

Two RGB LEDs show different colour changes.

The programme

The programme `16rgbled02` changes the three colours of each RGB LED cyclically with PWM signals. Different cycle times are used to create different, seemingly irregular colour changes.

The programme consists of six similar programme blocks, each of which controls one colour in one of the two RGB LEDs. They all start independent of each other at the moment when the user clicks on the green flag.

The programme uses twelve variables, which are all shown on the stage: **r1**, **g1** and **b1** contain the current colour values of the first RGB LED; **r2**, **g2** and **b2** contain the current colour values of the second RGB LED.

The variables **r1z**, **g1z** and **b1z** contain the waiting times between the individual switches of the three colours of the first RGB LED; **r2z**, **g2z** and **b2z** contain the corresponding values for the second RGB LED. These times can be changed interactively with a controller to alter the colour effects.

Each programme block changes the PWM signal of a colour component in ten steps from 0 to 500 and back again. Varying plays of colour are created by different activated colours.

The programme `16rgbled02` controls two RGB LEDs with PWM.

Day 17

Today in the advent calendar

- 1 blue LED with installed series resistor
- 1 cut-out template for the display instrument on the back

Analogue display with servo

The experiment of Day 17 uses the servo as analogue display instrument to indicate the position of a figure on the Scratch stage. The position is also shown by four coloured LEDs.

Components: 1 patch board, 1 servo, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 1 blue LED with installed series resistor, 8 GPIO connection cables, 4 pin strips (3 pins)

Cut out the scale on the cardboard back of the advent calendar and attach it to the servo so that it clicks into place at the servo housing. Cut out the pointer, as well. It is only attached to the servo once the programme moved the servo into its starting position.

Servo and four LEDs.

The pointer and the scale for the servo.

The programme

The programme `17servo02` allows the user to interactively move a figure on the Scratch stage. Its position is indicated by a servo and four LEDs.

The programme `17servo02` controls four LEDs and one servo based on the position of a figure.

A **Repeat continuously** loop checks all the time, which colours on the screen are touched by the figure and switches on the LED in the corresponding colours.

This figure is moved on the Scratch screen and its position is displayed.

The variable **s** is set to a value, which is a result of the y-position of the object divided by 1.8. The y-position of the figure can be a value between -180 and +180. However, this calculation is necessary because the values of the servo are between -100 and +100.

Start the programme by clicking on the green flag and drag the figure to the lower edge until 0 is shown in the field **Object1 y-position** and the blue LED lights up. Place the indicator onto the servo so that it points to 0 on the scale. The servo now shows the current position when you move the figure.

Day 18

Today in the advent calendar

- 4 GPIO connection cable

Rotating Christmas pyramid

The three LEDs are connected directly to the Raspberry Pi with GPIO connection cables. They are attached to the pyramid in the designated positions. Place the pyramid onto the servo arm so that it can rotate. If it does not stay firmly on the servo, remove the screw of the servo arm. Leave the arm on the servo, pierce the bottom of the pyramid with the screw and insert it into the servo arm. For the pyramid to be able to rotate freely, the LEDs are connected to the patch board with two connected GPIO connection cables. Once cable would be too short and impair the movement of the pyramid. All six cables from the LEDs are extended by a further six cables via two pin strips with three pins.

Components: 1 servo, 1 patch board, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 15 GPIO connection cables, 3 pin strips (3 pins)

Servo and three LEDs connected directly to the Raspberry Pi without patch board.

The programme

The programme `18pyramid01` consists of two independent programme blocks that control the rotation of the pyramid and the flashing of the LEDs. The speeds can be adjusted independently with controllers on the Scratch stage.

The programme `18pyramid01` controls the rotation and flashing of the pyramid.

The variable **rotate** contains the waiting time between two rotations in seconds. It can be adjusted with the controller to a minimum of one second, because the servo needs a certain time for its movements. The servo rotates between the positions 0, 100, 0 and -100 in an infinite loop.

The variable **flash** contains the time during which an LED is lit in tenths of a second. During this set time, the three LEDs light up one after another. The LEDs are toggled without waiting time. Thus, one LED is always switched on.

19. Day

Today in the advent calendar

- 1 GPIO connection cable

Controlling a rotating Christmas pyramid with a button

The experiment of Day 19 controls a flashing and rotating Christmas pyramid via button using a Python programme. The LEDs on the pyramid are again connected using two consecutive GPIO connection cables.

Components: 1 servo, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 1 button, 16 GPIO connection cables, 5 pin strips (3 pins)

LEDs on the rotating Christmas pyramid with a button on the patch board.

Pay attention to the correct assembly. The button and the servo use the same contact row for the +5 V signal.

The programme

The programme `19pyramid05.py` rotates the pyramid back and forth on the servo alternatingly in both directions. The LEDs light up consecutively as running light. Pressing the button reverses the direction of the running light.

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

LED = [[17, 22, 10],[10, 22, 17]]
x = 0
t1 = 8
servoPIN = 21

GPIO.setmode(GPIO.BCM)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(servoPIN, GPIO.OUT)

pwm = 2.5
a = 2.5
p = GPIO.PWM(servoPIN, 50)
p.start(pwm)

for i in LED[0]:
    GPIO.setup(i, GPIO.OUT)

try:
  while True:
    p.ChangeDutyCycle(pwm)
    pwm += a
    if pwm==12.5 or pwm==2.5:
      a = -a
    time.sleep(0.2)
    p.ChangeDutyCycle(0)
    for i in LED[x]:
      GPIO.output(i, True)
      time.sleep(0.2)
      GPIO.output(i, False)
      if GPIO.input(t1) == True:
        x = 1-x

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

This is how the programme works

```
LED = [[17, 22, 10],[10, 22, 17]]
x = 0
```

The pins for the three LEDs are defined as nested list. The first list element is another list with the three pin numbers, the second list element is a list with the pin numbers in reverse order. The variable `x` is later used to select one of the two lists. Counting of list elements always starts with 0. Therefore, the two elements of this list have the numbers `LED[0]` and `LED[1]`.

```
for i in LED[0]:
    GPIO.setup(i, GPIO.OUT)
```

The three GPIO pins are initialised with the list `LED[0]`. Since the two lists contain the same pin numbers, the other list can be used in exactly the same way.

```
    p.ChangeDutyCycle(pwm)
    pwm += a
```

Inside the infinite loop, the servo is assigned the current `pwm` value during each loop iteration. This value is increased by the variable `a`, which was defined earlier as `2.5`.

```
    if pwm==12.5 or pwm==2.5:
      a = -a
```

Once the servo has reached on the two positions, the sign of the variable `a` is changed so that the servo rotates in the other direction during the next loop iteration.

```
    time.sleep(0.2)
    p.ChangeDutyCycle(0)
```

After a waiting time of 0.2 seconds, the servo receives a PWM signal with the value 0 to avoid the familiar vibration.

```
    for i in LED[x]:
      GPIO.output(i, True)
      time.sleep(0.2)
      GPIO.output(i, False)
      if GPIO.input(t1) == True:
        x = 1-x
```

This starts a loop that controls the three LEDs. It switches each of them on consecutively for 0.2 seconds. The button is queried during each iteration of this loop. If it is pressed, the variable `x` changes from 0 to 1 or the other way around. This activates the respective other list of the LEDs for the running light and the light direction is reversed.

After one running light cycle, the infinite loop starts again. It rotates the servo in one step and starts another running light cycle.

Pressing the shortcut [Ctrl]+[C] on the keyboard stops the PWM signal and the PWM pins are reset.

Day 20

Today in the advent calendar

- 1 GPIO connection cable

Controlling a rotating Christmas pyramid with an app

Today, the Christmas pyramid is controlled with a mobile app.

Components: 1 servo, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 15 GPIO connection cables, 3 pin strips (3 pins)

Servo and three LEDs connected directly to the Raspberry Pi without patch board.

The programme

The movement of the pyramid and the toggling of the LEDs are controlled by an app. There will be one button for rotation to the right and one button for rotation to the left. There will also be one button each for the running light and the flashing of the LEDs. One Python script is used for each these four functions. For rotations to the left, use the programme `20servo_le.py`:

```
#!/usr/bin/python
#20servo_le.py
import RPi.GPIO as GPIO
import time

servoPIN = 21
pwm = 12.5
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)
p = GPIO.PWM(servoPIN, 50)
p.start(pwm)
time.sleep(0.5)
p.ChangeDutyCycle(0)
p.stop()
GPIO.cleanup()
```

For rotations to the right, use the programme `20servo_ri.py`:

```
#!/usr/bin/python
#20servo_ri.py
import RPi.GPIO as GPIO
import time

servoPIN = 21
pwm = 2.5

GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)
p = GPIO.PWM(servoPIN, 50)
p.start(pwm)
time.sleep(0.5)
p.ChangeDutyCycle(0)
p.stop()
GPIO.cleanup()
```

To control the running light, use programme `20pyramid_runninglight.py`:

```
#!/usr/bin/python
#20pyramid_runninglight.py
import RPi.GPIO as GPIO
import time

LED = [17, 22, 10]
GPIO.setmode(GPIO.BCM)

for i in LED:
  GPIO.setup(i, GPIO.OUT)

for j in range(3):
  for i in LED:
    GPIO.output(i, True)
    time.sleep(0.2)
    GPIO.output(i, False)

GPIO.cleanup()
```

To switch on the flashing lights, use programme `20pyramid_flash.py`:

```
#!/usr/bin/python
#20pyramide_flash.py
import RPi.GPIO as GPIO
import time

LED = [17, 22, 10]
GPIO.setmode(GPIO.BCM)

for i in LED:
  GPIO.setup(i, GPIO.OUT)

for j in range(3):
  for i in LED:
    GPIO.output(i, True)
  time.sleep(0.2)
  for i in LED:
    GPIO.output(i, False)
  time.sleep(0.2)

GPIO.cleanup()
```

The app

If you have not installed the web server and PHP5, please follow the steps outline in Day 7.

To implement the app, again you need to go through the same four steps as on Day 7.

Step 1-App: Copying the template into the web server directory

First, you need to copy the template `app_template` into a directory on the web server. To do this, follow Step 1 for the app on Day 7. This time, name your directory `day20` instead of `day07`.

Step 2-App: Customising the template

Open the file `index.php` from the directory `day20` by entering `nano index.php` into the LXTerminal. Add the required buttons to the empty block `form`:

```
<form action="index.php" method="post">
 <input name="servo_left" id="servo_left" type="submit"
  value="Pyramid left" style="background-color: #4180C5">
 <input name="servo_right" id="servo_right" type="submit"
  value="Pyramid right" style="background-color: #4180C5">
 <input name="runninglight" id="runninglight" type="submit"
  value="Running light" style="background-color: #4180C5">
 <input name="flash" id="flash" type="submit" value="Flashing"
  style="background-color: #4180C5">
</form>
```

This will create four blue buttons with the texts **Pyramid left**, **Pyramid right**, **Running light** and **Flashing**. Save the file with [Ctrl]+[O] and close the editor with [Ctrl]+[X]. Then, open the file `index.php` again in the browser using **http://localhost/day20/index.php**.

The template now only needs to be linked to the two Python scripts. To do this, change the beginning of the file `index.php` as follows:

```
<head>
 <?php
  if ($_POST["servo_left"]) {
   echo shell_exec('sudo /var/www/html/day20/20servo_le.py');
  } elseif ($_POST["servo_right"]) {
   echo shell_exec('sudo /var/www/html/day20/20servo_re.py');
  } elseif ($_POST["runninglight"]) {
echo shell_exec('sudo /var/www/html/day20/20pyramid_runninglight.py');
  } elseif ($_POST["flash"]) {
   echo shell_exec('sudo /var/www/html/day20/20pyramid_flash.py')
  }
 ?>
<title>Conrad Raspberry Pi Advent Calendar</title>
```

The mobile webpage is now complete and you can assign the authorisations.

Step 3: Defining authorisations

Open the LXTerminal again and go to the directory of Day 20 with `cd /var/www/html/day20`. Provide the scripts with the execution authorisation:

```
chmod a+x 20*.py
```

Now you need to authorise the web server to run this script. Use the following command for this:

```
sudo visudo
```

Now you can control the pyramid and the ELDs with your smartphone.

Insert the following lines:

```
www-data ALL=(ALL) NOPASSWD: /var/www/html/day20/20servo_li.py
www-data ALL=(ALL) NOPASSWD: /var/www/html/day20/20servo_ri.py
www-data ALL=(ALL) NOPASSWD:   /var/www/html/day20/20pyramid_runninglight.py
www-data ALL=(ALL) NOPASSWD:  /var/www/html/day20/20pyramid_flash.py
```

Save the file with [Ctrl]+[O] and close it with [Ctrl]+[X].

Step 4: Testing the app

You did it! Now you can test the app on your smartphone. Start the browser in your smartphone, open the page with **http://<IP-Adresse Raspberry Pi/day20/index.php** and now you can control the pyramid.

Day 21

Today in the advent calendar

• 1 20-Mohm resistor

Christmas pyramid controlled with modelling clay

In the experiment of Day 21, two modelling clay contacts control the flashing LEDs on the Christmas pyramid using a Python programme. The pyramid does not rotate this time. Since the pyramid does not rotate today, single GPIO cables are sufficient for the LEDs. It is not necessary to extent them with a second cable. Thus, the existing pin strips can be used for the two clay contacts.

Components: 1 patch board, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 2 20-Mohm resistors (red-black-blue), 11 GPIO connection cables, 5 pin strips (3 pins), two modelling clay contacts

Two clay contacts control three LEDs on the Christmas pyramid.

The programme

The programme `21clay03.py` shows how the clay contacts are activated with Python. When touching one clay contact, the LEDs flash as running light. When touching the other contact, all LEDs flash at the same time.

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

LED = [17, 22, 10]
k1 = 13
k2 = 20

GPIO.setmode(GPIO.BCM)
GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)

for i in LED:
    GPIO.setup(i, GPIO.OUT)

try:
  while True:
    while GPIO.input(k1) == False:
      for i in LED:
        GPIO.output(i, True)
        time.sleep(0.1)
        GPIO.output(i, False)

    while GPIO.input(k2) == False:
      for i in LED:
        GPIO.output(i, True)
      time.sleep(0.1)
      for i in LED:
        GPIO.output(i, False)
      time.sleep(0.1)

except KeyboardInterrupt:
  GPIO.cleanup()
```

This is how the programme works

```
LED = [17, 22, 10]
k1 = 13
k2 = 20
```

The GPIO pins for the LEDs are saves as lists again; the pins of the clay contacts are saves as variables `k1` und `k2`.

```
    while GPIO.input(k1) == False:
      for i in LED:
        GPIO.output(i, True)
        time.sleep(0.1)
        GPIO.output(i, False)
```

Two `while` loops run inside the infinite loop. Each of them runs as long as one of the two clay contacts is touched. Inside the first `while` loop, a `for` loop runs that switches on the three LEDs consecutively for 0.1 seconds each.

```
    while GPIO.input(k2) == False:
      for i in LED:
        GPIO.output(i, True)
      time.sleep(0.1)
      for i in LED:
        GPIO.output(i, False)
      time.sleep(0.1)
```

If the other clay contact is touched, all three LEDs are switched on in direct succession and they are all switched off again after waiting time of 0.1 seconds. This will look as if all three LEDs flash at the same time.

Day 22

Today in the advent calendar

· 1 pin strip (3 pins)

Controlling a rotating Christmas pyramid with modelling clay

The experiment of Day 22 is based on the experiment on Day 18. The difference is that the LEDs well only flash and the Pyramid will only rotate if the clay contact is touched. The LEDs on the pyramid are again connected using two consecutive GPIO connection cables.

Components: 1 servo, 1 patch board, 1 red LED with installed series resistor, 1 yellow LED with installed series resistor, 1 green LED with installed series resistor, 1 20 Mohm resistor (red-black-blue), 17 GPIO connection cables, 6 pin strips (3 pins), 1 modelling clay contact

Controlling the Christmas pyramid with a clay contact.

The programme

Similar to the programme on Day 18, the programme `22pyramid02` consists of two independent blocks that control the rotation of the pyramid and the flashing effects of the LEDs. Instead of infinite loops, the programme contains sensor queries. First, a block **wait until...** waits for the value of the GPIO pin 8 become 0 due to the modelling clay contact being touched. After that, a **repeat until...** loop is executed until the value of the GPIO pin 8 changes to 1 due to the contact to the clay being interrupted.

The programme 22pyramid02 controls the pyramid with a clay contact.

Due to the relatively slow rotation of the servo, the LEDs will stop flashing almost immediately when the clay contact is no longer touched but the servo will continue to rotate for a moment.

This programme also has a block that sends the command **servo18stop** and, therefore, switches off the GPIO pin when the [spacebar] is pressed. The servo stops and does not vibrate.

Day 23

Today in the advent calendar
• 1 RGB LED with installed series resistor

Plays of RGB colour with Python
The experiment of Day 23 makes three RGB LEDs light up automatically in different colours using PWM signals.

Components: 1 patch board, 3 RGB LEDs with installed series resistors, 10 GPIO connection cables, 5 pin strips (3 pins)

Three RGB LEDs with series resistors.

The programme
The programme `23rgbled03.py` generates effects of changing colour in the RGB LEDs using the HSV colour system.

HSV and RGB colour system
The RGB colour system has been used in all programmes so far and describes colours in three components - red, green and blue - that are mixed together. For humans, it is relatively difficult to imagine a mixed colour. In contrast to that, the HSV colour system describes the colours with the parameters H = Hue, S = Saturation and V = Value (brightness).
Simply by changing the H-value, all colours of the colour spectrum can be described in full intensity if the other two values are set to maximum.

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

GPIO.setmode(GPIO.BCM)
LED1 = [21, 20, 16]
LED2 = [12, 7, 8]
LED3 = [25, 24, 23]
p1 = [0, 0, 0]
p2 = [0, 0, 0]
p3 = [0, 0, 0]

for i in range(3):
  GPIO.setup(LED1[i], GPIO.OUT)
  GPIO.setup(LED2[i], GPIO.OUT)
  GPIO.setup(LED3[i], GPIO.OUT)

for i in range(3):
  p1[i] = GPIO.PWM(LED1[i], 50)
  p1[i].start(0)
  p2[i] = GPIO.PWM(LED2[i], 50)
  p2[i].start(0)
  p3[i] = GPIO.PWM(LED3[i], 50)
  p3[i].start(0)

try:
  while True:
    for i in range(100):
      rgb1 = colorsys.hsv_to_rgb(i/100, 1, 1)
      x2 = i+20
      if x2 > 100:
        x2 -= 100
      rgb2 = colorsys.hsv_to_rgb(x2/100, 1, 1)
      x3 = i+40
      if x3 > 100:
        x3 -= 100
      rgb3 = colorsys.hsv_to_rgb(x3/100, 1, 1)
      for j in range(3):
        p1[j].ChangeDutyCycle(rgb1[j])
        p2[j].ChangeDutyCycle(rgb2[j])
        p3[j].ChangeDutyCycle(rgb3[j])
      time.sleep(0.1)

except KeyboardInterrupt:
  for i in range(3):
    p1[i].stop()
    p2[i].stop()
    p3[i].stop()
  GPIO.cleanup()
```

This is how the programme works

```
import colorsys
```

In addition to the libraries that are already known, the library `colorsys` is imported to convert between the HSV and the RGB colour system.

```
LED1 = [21, 20, 16]
LED2 = [12, 7, 8]
LED3 = [25, 24, 23]
```

The pin numbers of all three RGB LEDs are then saved in their own list.

```
p1 = [0, 0, 0]
p2 = [0, 0, 0]
p3 = [0, 0, 0]
```

The nine PWM signals of the three RGB LEDs are also arranged in three lists.

```
for i in range(3):
  GPIO.setup(LED1[i], GPIO.OUT)
  GPIO.setup(LED2[i], GPIO.OUT)
  GPIO.setup(LED3[i], GPIO.OUT)
```

A loop defines all used GPIO pins as outputs.

```
for i in range(3):
  p1[i] = GPIO.PWM(LED1[i], 50)
  p1[i].start(0)
  p2[i] = GPIO.PWM(LED2[i], 50)
  p2[i].start(0)
  p3[i] = GPIO.PWM(LED3[i], 50)
  p3[i].start(0)
```

Another loop sets up the nine PWM signals and assigns them all a starting value of 0. All RGB LEDS are switched off.

The HSV colour system normally uses values between 0 and 100 for the colour scale. In contrast, the RGB system uses values between 0 and 255. In Python, both colour systems use values between 0.0 and 1.0.

```
    for i in range(100):
      rgb1 = colorsys.hsv_to_rgb(i/100, 1, 1)
```

To convert the colour systems, a loop runs from 0 to 99. Inside this loop, the RGB values of the first RGB LED are re-calculated with every loop iteration by converting the loop counter as HSV value into an RGB value using the function `colorsys.hsv_to_rgb()`. This function automatically creates a list of three values saved in the variable `rgb1`.

```
      x2 = i+20
      if x2 > 100:
        x2 -= 100
      rgb2 = colorsys.hsv_to_rgb(x2/100, 1, 1)
```

The second RGB LED receives an HSV value that is shifted by 20 compared to the first LED. This value is converted using the same formula and saved in the variable `rgb2`. Once the value created by adding to the loop counter exceeds the limit of 100, a value of 100 will automatically subtracted so that it starts again with 0.

```
      x3 = i+40
      if x3 > 100:
        x3 -= 100
      rgb3 = colorsys.hsv_to_rgb(x3/100, 1, 1)
```

In the same way, the colour value of the third RGB LED is calculated by adding 40 to the HSV value of the first LED. The colour spectrum of the second and third RGB LED, therefore, lag behind the first LED.

```
      for j in range(3):
        p1[j].ChangeDutyCycle(rgb1[j])
        p2[j].ChangeDutyCycle(rgb2[j])
        p3[j].ChangeDutyCycle(rgb3[j])
      time.sleep(0.1)
```

A loop then sets the PWM signals of the three RGB LEDs to the calculated RGB values and waits for 0.1 seconds before the next colours are calculated.

```
except KeyboardInterrupt:
  for i in range(3):
    p1[i].stop()
    p2[i].stop()
    p3[i].stop()
  GPIO.cleanup()
```

If the user ends the programme by pressing the shortcut [Ctrl]+[C] on the keyboard, all nine PWM signals are stopped and the GPIO pins are reset.

Day 24

Today in the advent calendar

· 1 download code

· 1 cut-out template for the display instrument on the back

Clock for waiting time until the next advent calendar.

A clock displays the waiting time until the next calendar and plays a melody fitting for the time of year. To do this, download the songs from **www.buch.cd** using the download code find behind today's door. Unpack the zip archive into the home directory `/home/pi`.

Components: 1 servo, 1 pin strip (3 pins), 3 GPIO connection cables, download code for music

On the back of the advent calendar, you can find a scale with twelve months, which you can attach to the servo.

Servo to control the clock.

The scale for the twelve months.

The programme

The programme `24clock01.py` uses the servo to show how many months you still have to wait until the next advent calendar and plays a song.

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

servoPIN = 21
GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)
p = GPIO.PWM(servoPIN, 50)
p.start(0)

time = time.localtime()
m = time.tm_mon
pwm = (m/12*10)+2.5
p.ChangeDutyCycle(pwm)
time.sleep(0.5)
p.ChangeDutyCycle(0)

if m<4:
    subprocess.Popen(["omxplayer", "lied1.mp3"])
elif m<7:
    subprocess.Popen(["omxplayer", "lied2.mp3"])
elif m<10:
    subprocess.Popen(["omxplayer", "lied3.mp3"])
elif m<12:
    subprocess.Popen(["omxplayer", "lied4.mp3"])
else:
    subprocess.Popen(["omxplayer", "lied5.mp3"])

p.stop()
GPIO.cleanup()
```

This is how the programme works

```
import subprocess
```

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

As you already know, the GPIO pin is defined as output and a PWM signal is started.

```
time = time.localtime()
```

The current time is saved in the object `time`. To do this, the function `time.localtime()` from the `time` library is used. The result is a data structure that consists of several individual values.

```
m = time.tm_mon
```

The month is extracted from the object time and saved in the variable `m`. The month is a number between 1 (January) and 12 (December).

```
pwm = (m/12*10)+2.5
p.ChangeDutyCycle(pwm)
```

Using the value for the month, the corresponding PWM value in the rotatory range of the servo is calculated and the servo is rotated to this value.

```
time.sleep(0.5)
p.ChangeDutyCycle(0)
```

After a waiting time of half a second, the servo is assigned the PWM value 0 to prevent vibrations. It stays on the position calculated before.

```
if m<4:
    subprocess.Popen(["omxplayer", "lied1.mp3"])
```

If the current month is less than 4 (January to March), the melody `lied1.mp3` is played. The function `subprocess.Popen()` is used to this, which opens the command line based media player `omxplayer` in this case.

```
elif m<7:
    subprocess.Popen(["omxplayer", "lied2.mp3"])
elif m<10:
    subprocess.Popen(["omxplayer", "lied3.mp3"])
elif m<12:
    subprocess.Popen(["omxplayer", "lied4.mp3"])
```

The conditions behind `elif` are evaluated, if none of the conditions before gave the value `True`. In this way, any number of conditionals can be nested behind each other. Depending on the current month (April to June, July to September, October to November), different songs are played.

```
else:
    subprocess.Popen(["omxplayer", "lied5.mp3"])
```

If none of the conditions returns the value `True`, the current month is December. In this case, the melody `lied5.mp3` is played.

At the end, the PWM signal is stopped and the GPIO pins are reset.

Merry Christmas!