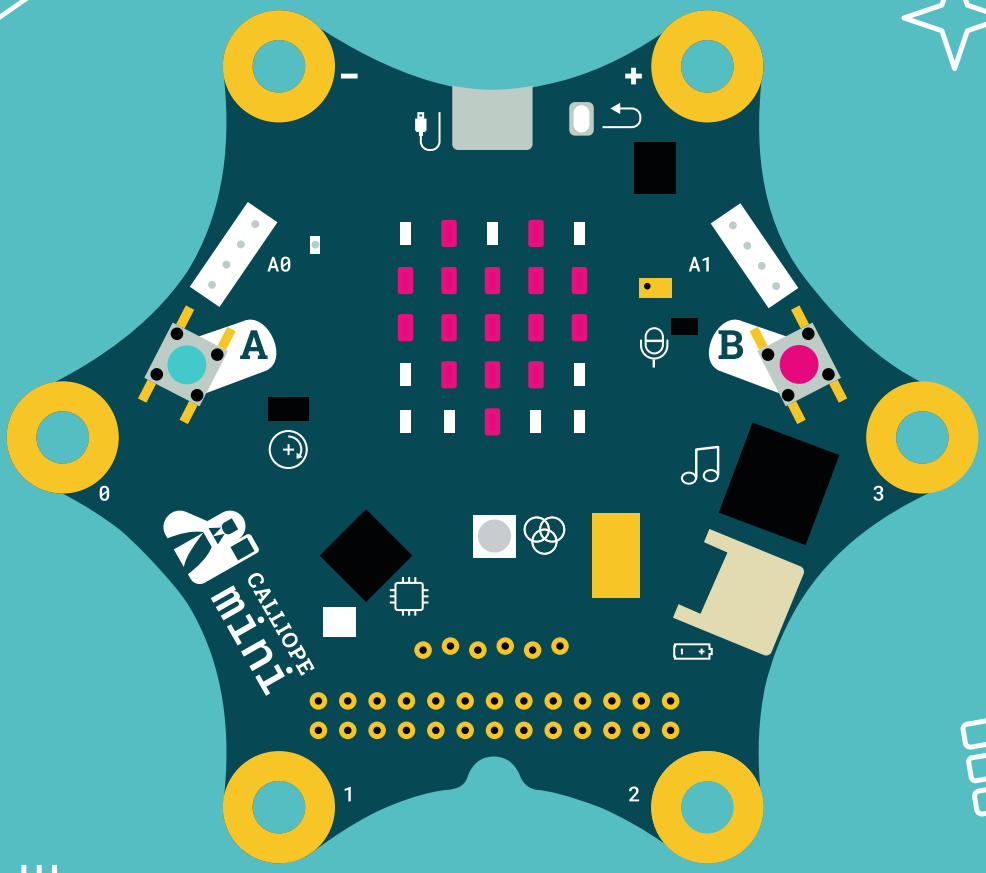


Coding with the Calliope mini

Teaching material for use from grade 3

CALLIOPE

Coding with the Calliope mini

Programming in primary school

Teaching material for use from grade 3

Authors: Michael Abend (Glossary, the *Calliope mini* and the Nim game, the *Calliope mini* as a spelling trainer, the *Calliope mini* as a stopwatch and countdown counter),
Kirstin Gramowski (Morse code with the *Calliope mini*, the *Calliope mini* as a 1 x 1 mental arithmetic trainer, the *Calliope mini* as a random generator, determine neighbouring numbers with the *Calliope mini*),
Lars Pelz (the *Calliope mini* as a metronome, the *Calliope mini* as a mini piano, the *Calliope mini* as an automatic bicycle rear light, generating image impulses and stimulus words with the *Calliope mini*),
Bernd Poloczek (Learning map, epilogue (experiences & conditions for success))

Consultants: Michael Abend, Kirstin Gramowski, Lars Pelz, Bernd Poloczek

Editor: Patrizia Schwarzer

Illustration: Calliope gGmbH, Berlin: Cover, p. 3, p. 5, p. 62/63, back cover zweiband.media GmbH, Berlin: all remaining illustrations on the inside

Photo: Sibylle Baier, Berlin: p. 64

Cover design: original cover design made by COSAKitchen, Corinna Babylon, Berlin
English cover design made by Calliope gGmbH, Berlin

Layout, graphics and technical implementation: zweiband.media GmbH, Berlin

www.cornelsen.de
www.cornelsen.de/calliope

2nd edition Print 2018

All prints in this edition are unchanged in terms of content and can be used in parallel in lessons.

Translated by Calliope gGmbH based on the original title “Coden mit dem *Calliope mini*” 2019 Calliope gGmbH, Berlin

ISBN 978-3-06-600012-2

Print: Athesiadruck GmbH

PEFC zertifiziert
Dieses Produkt stammt aus nachhaltig bewirtschafteten Wäldern und kontrollierten Quellen.
www.pefc.de
PEFC
PEFC/18-31-166

Advice: The NEPO Editor is constantly being improved and further developed. It may happen that some blocks look a little different. However, the functions remain the same, so that you can realise all the shown exercises.

Table of contents

Dear teachers, dear educational professionals,

The writing of computer programs, so-called coding, is still a relatively new trend at school, which offers the emerging generation the opportunity to develop technological knowledge and to use it for the action-oriented discovery of the world.

Cornelsen Verlag supports the initiative of *Calliope gGmbH*, with the help of which every schoolchild in Germany from the 3rd grade onwards should be able to have playful access to the digital world via a small programmable microcontroller, the *Calliope mini*.

*"With the **Calliope mini** even primary school children can learn creatively and playfully how the digital world works." (Gesche Joost, professor for design research at the Berlin University of the Arts and co-founder of Calliope gGmbH)*

At first glance, programming appears not to be a trivial necessity, even at primary school. Neither the primary school syllabus nor other (educational) regulations make the subject binding. However, if programming is not viewed as an end in itself, but is rather placed in a more general context, a wide range of points of contact opens up, especially for issues relevant to primary schools.

One aspect is the importance of media literacy. This encompasses much more than the pure operating competence of digital devices. Therefore, not only working with media, but also learning about media plays a major role. A development of competence in the area of coding also contributes to a basic understanding of the digital world that surrounds us, which allows us to demystify this technological "black box". Instead of remaining in a receptive stage, an initial understanding of programming enables critical, reflective judgment, such as making use of technology (e.g. processing routines) and initiating the confident handling of technological processes. In addition, participation in the digital society is made possible.

During the course, programming opens up a range of skills that are specifically developed: Coding …

- develops exploratory and discovery learning through a playful approach and fault-tolerant work.
- develops abstract capability, modelling and problem-solving skills as well as analytical thinking and clear structuring.
- develops higher level skills: Independence, initiative and the creation of own hypotheses.
- develops constructive approaches and creative-productive solution processes.
- contributes to a high level of student activation: Coding creates opportunities for action and discussion.
- It develops natural differentiation by means of many correct paths to the (self-established) goal and offers a wide range of design options.

As the didactic cooperation partner of *Calliope gGmbH*, at Cornelsen Verlag we group together the above-mentioned points and transfer them in this teaching material into specific examples with reference to the syllabus.

What to expect:

- 11 coding examples with the *Calliope mini* tailored for the contents of the syllabus for the subjects general knowledge, German and mathematics in primary school from grade 3
- Step-by-step instructions for programming beginners - You can create your own programs without any previous programming knowledge and systematically build up your coding skills.
- To understand coding as a tool for problem solving and to recognise that it is never just an end in itself
- Selected examples that offer a different approach to syllabus topics (including the example of Morse code with the *Calliope mini*) - which cannot be created with conventional media (books, notebooks, ...) or cannot be created very quickly (e.g. the *randomly controlled 1 x 1 mental arithmetic trainer*) - or show specific application references (e.g. the *Calliope mini as an automatic bicycle rear light*)

Try it out for yourself and later in your class!

How does a computer “think”?

Fortunately, a computer can not (yet) think for itself. It consists of an electronic circuit (processor) that can only calculate with ones and zeros, but it does so at an almost unimaginable speed. What and how the computer should calculate must be “taught” beforehand. This activity is called programming or coding.
When people have tasks to accomplish (in this context we are speaking about problems) that are time-consuming and labour-intensive to solve, they can transfer the processing of such problems to a computer. To do so, the procedure for solving the problem must be formulated as a series of clear instructions. Such a sequence is called an “algorithm” in computer science.

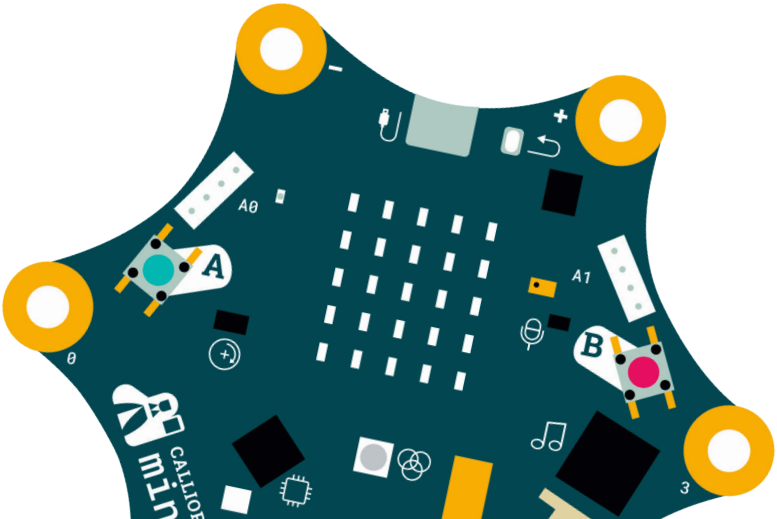
An algorithm can be written in a programming language. This is a kind of interim solution, a language that can still be read by humans, but can also be processed by the computer. The resulting program adjusts the electronic circuitry of the computer so that it carries out exactly the work steps that were previously described in the algorithm. The structure of an algorithm mostly follows the basic principle 1) input (information that reaches the computer from its environment), 2) processing (according to the rules of the algorithm, whereby new information arises) and 3) output (the newly created information via various means).

Coding with a graphic programming language using the example of *NEPO®*

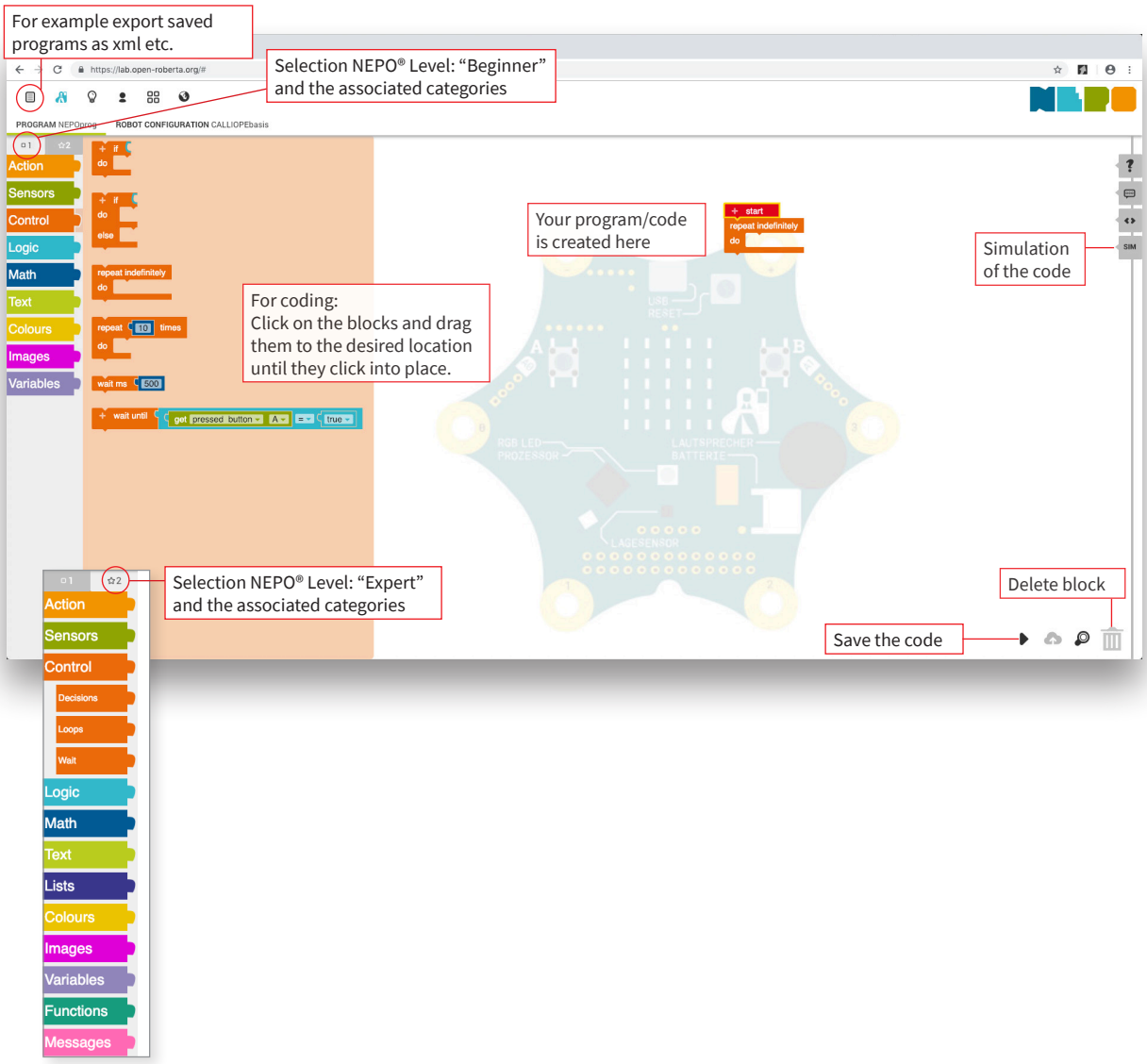
Programming languages basically consist of commands, decisions and repetitions. A sequence of these instructions stored in the computer is called a program. Programs clearly and precisely describe what the computer running a program should do to solve a problem associated with it. There are different programming languages (text-based and graphic) with which programs are written. The term coding refers to the activity of creating a program i.e. the coding of a sequence of instructions to solve a problem for the computer.
In contrast to text-based programming languages, which require a high level of abstract capacity, graphic programming languages offer a simple introduction to coding. With them, the basic concepts of computer programming can be learned in a playful and explorative way. Prefabricated code modules (blocks) are nested like puzzle pieces in order to create programs. So-called editors (interfaces within which programming is carried out) enable a quick and comprehensive overview of the commands and structures available.

Another advantage of graphic programming is that program elements (blocks) can only be put together where they are allowed to according to the underlying system of rules. This avoids syntax errors, which can cause great frustration when coding in text-based programming languages. Graphic programming languages are widely represented today. At their core, they always work according to the same principles. Once you have understood the basic idea of coding, i.e. the formulation of a specific sequence of instructions as a problem-solving tool, you can immediately start using any graphic programming language.
In the following examples, the graphic programming language *NEPO®* is used in the *Open Roberta Lab®* from the Fraunhofer Institute. The internet-based editor of this open source platform does not require any installation, just an internet connection and an up-to-date browser.
In the following you will get to know the editor *Open Roberta Lab®* and the programming language *NEPO®*:

- First go to the calliope.cc/en/coding/editors website in a current browser. Then click on the editor *Open Roberta Lab®*.
- Inside the *Open Roberta Lab®* select the system *Calliope*.
- After selecting the system, you can start programming with *NEPO®*.



The *Open Roberta Lab*® and the programming language *NEPO*®



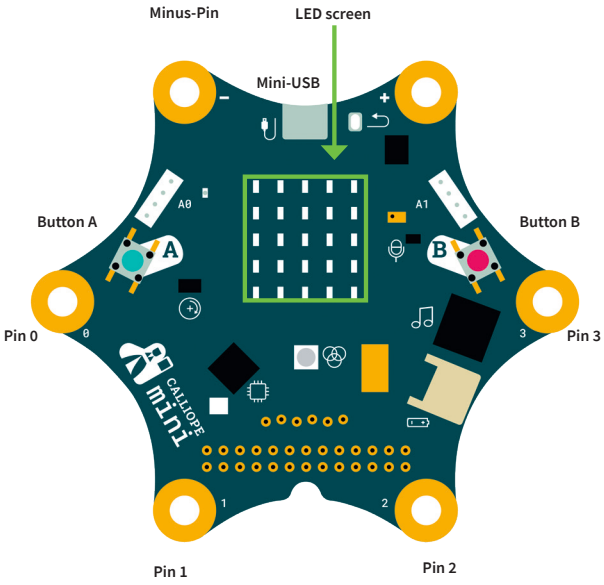
Procedure for the following examples as well as general tips for coding:

1. Set beginner or expert level as indicated in the example. The two modes do not refer to the level the difficulty, but rather contain more or less blocks.
2. When you click on the *NEPO*® categories, the associated blocks appear. An explanation of the respective block appears when the mouse pointer hovers over it.
3. Select the required block and, holding down the left mouse button, drag it under the red “Start” block. You place all other blocks in the appropriate places within the code so that the new block clicks into the existing structure.
4. Clicking the right mouse button offers you a number of further options (e.g. copying, deleting a block)
5. A comprehensive explanation of the blocks and their possible uses can be found at: https://jira.iais.fraunhofer.de/wiki/display/ORInfo/Programmieren+Calliope+mini
6. The program can be executed in simulation mode on the screen. You can check on the screen whether your program is running as you want it to. To run this simulation, click on “SIM” in the right-hand column.
7. Saving the code: To transfer the program created to the *Calliope mini* and then test it, click on the “Play” button at the bottom right and follow the instructions on the screen.
8. Create a profile if you want to make your programs accessible from anywhere, continue working on programs or share them with others.

Program the *Calliope mini* with *NEPO*®

So-called microcontrollers or microcomputers are suitable for getting started with programming. These have very simple input and output options (e.g. a few buttons instead of a keyboard, a few LEDs instead of a high-resolution display).
The *Calliope mini* is such a microcontroller that can be controlled, among other things, by means of a graphic programming language and works according to the principles of an algorithm. The *Calliope mini* has sensors for input, for example, which can measure the light intensity, its position in space or the strength of the earth's magnetic field. Buttons can also be pressed to generate inputs. The *Calliope mini* has, for example, an LED screen, an RGB LED and a loudspeaker available for output. Furthermore, motors can be connected whose speed and direction of rotation can be controlled by the processor in the *Calliope mini*.
If you have coded your first own program with *NEPO*®, you can transfer it to your *Calliope mini* and run it there.
All examples in this book combine both components (*NEPO*® and *Calliope mini*) and show various ways in which you can program the *Calliope mini* so that it can be used effectively in your specialist teaching.

The most important functions of the Calliope mini



Further information about functionalities and sensors can be found at: **calliope.cc/en/calliope-mini/features**

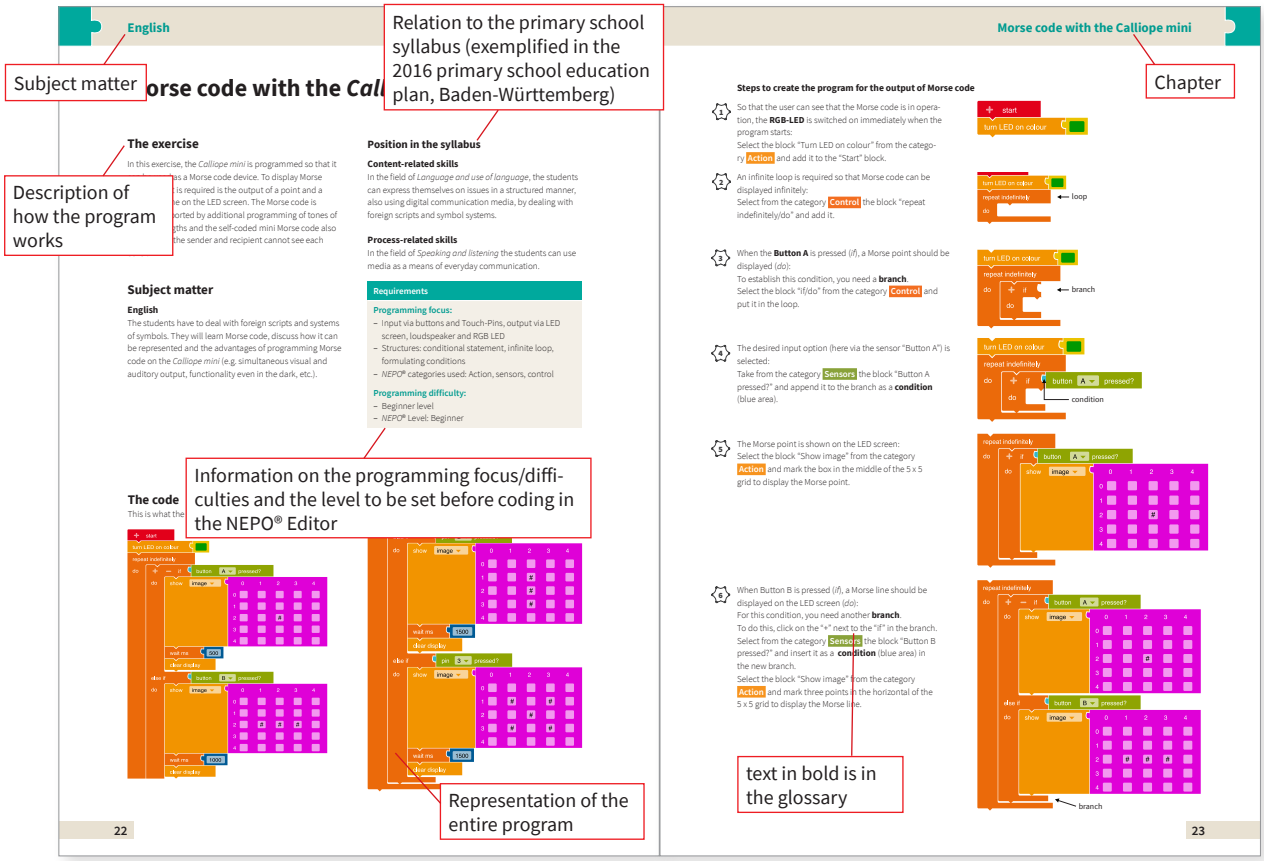
You will need the following to test the examples

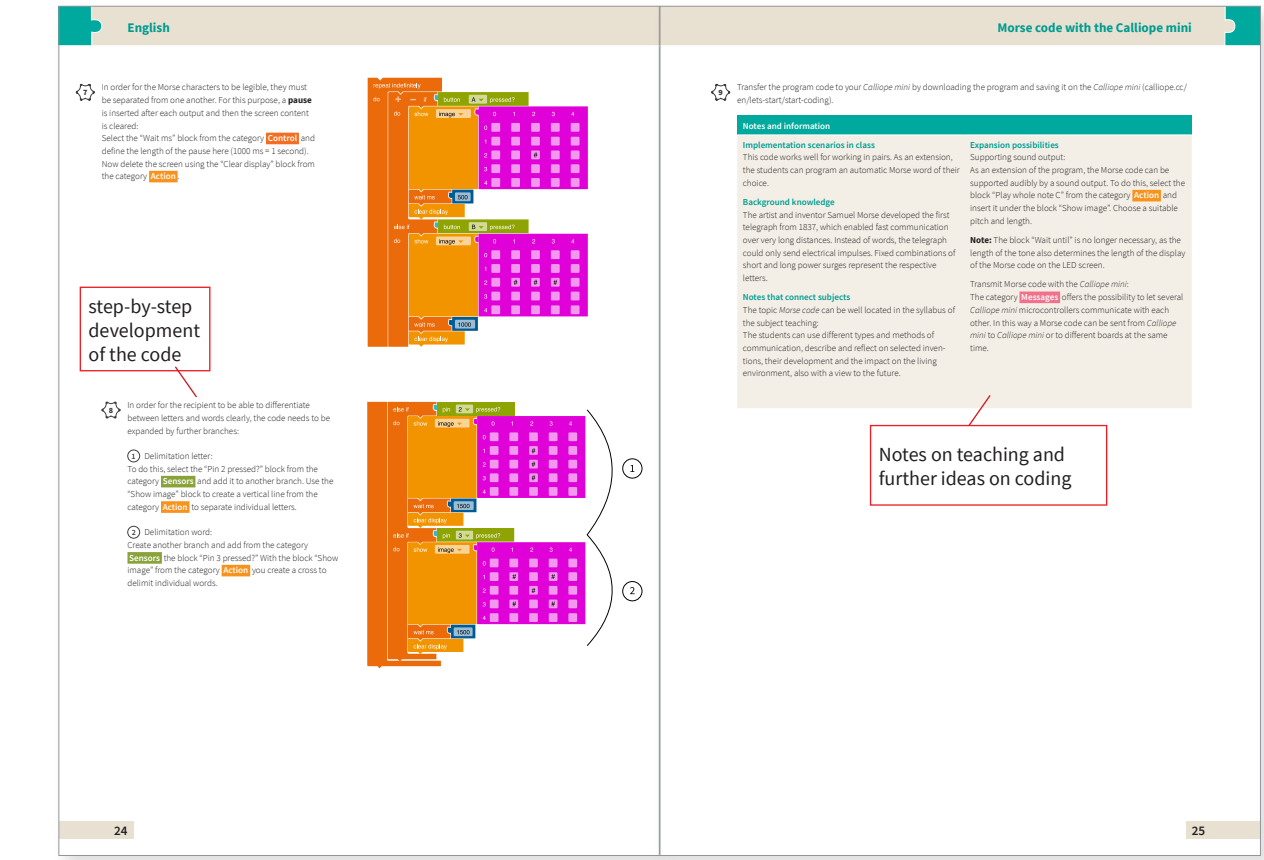
- A device with internet capability (e.g. laptop, PC)
- The web-based Editor *NEPO*® from *Open Roberta Lab*® for the programming, which can be found at **calliope.cc/en/coding/editors**
- A battery pack, to supply the *Calliope mini* with electricity
- A connection cable (USB-mini to USB) to transfer the code to the *Calliope mini*
- Further assistance can be found at **calliope.cc/en/lets-start/intro**

How to transfer your program to the *Calliope mini*:

- Connect the *Calliope mini* to the computer using a USB to USB mini cable.
- The *Calliope mini* appears as a storage device (USB drive).
- Save the program file downloaded from *NEPO*® either directly on the *Calliope mini* or move the file from the local storage location to the "MINI" drive.
- While the program is being transferred, the light to the left of the USB port flashes.
- After the transfer is complete, the light lights up continuously as long as the circuit board is supplied with power.
- The *Calliope mini* is now ready to execute the program that has been transferred. With the *Calliope 2016* you must first press the reset button.
- Note: The *Calliope mini* saves only one program at a time. If a new program is transmitted, it will overwrite the program that has already been loaded.
- Start the program on the *Calliope mini* by triggering the input specified in your programming, for example by pressing the A button.
- You can then see the result of your program on the *Calliope mini*.

Structure of the examples





The *Calliope mini* as an automatic bicycle rear light

The exercise

The *Calliope mini* is programmed as an automatic bicycle rear light. The light should turn on when it gets dark and turn off again as soon as it is bright enough. To do this, the light sensor built into the *Calliope mini* is queried.

The light sensor of the *Calliope mini* converts the incidental light into a numerical value so that the respective light levels can be differentiated by the *Calliope mini* processor. The lower the incidence of light, the lower the associated numerical value. The programmer must define the numerical values determined by the sensor for which the boundary between "dark" and "light" is drawn.

Subject matter

General knowledge

When programming the automatic bicycle light, the students learn that the *Calliope mini* can measure the strength of the incidental light. They find that you can vary the light intensity by putting the *Calliope mini* in the shade.

Position in the syllabus

Content-related skills

In the field of *Space and mobility*, the students can describe and implement requirements for the safe participation in traffic, as well as checking and maintaining their bicycle with regard to its traffic safety.

Process-related skills

In the field of *Communication and understanding*, the students can express ideas, learning and solution paths and knowledge gained (e.g. when planning and building technical products or when comparing means of transport).

Requirements

Programming focus:

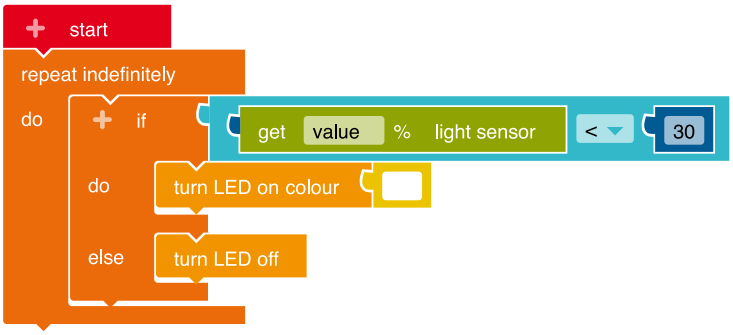
- Query light sensor
- Program structures used: Infinite loop, branch, condition, command
- *NEPO*® categories used: Action, control, sensors, mathematics, logic

Programming difficulty:

- Beginner level
- *NEPO*® Level: Beginner

The code

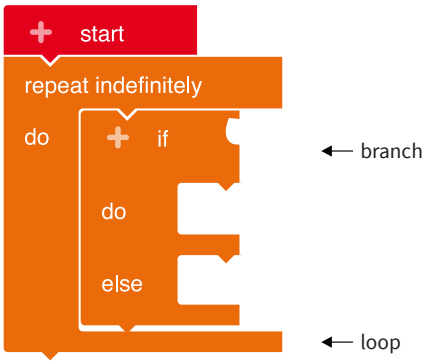
This is what the code of the finished program looks like. It is defined step by step below.



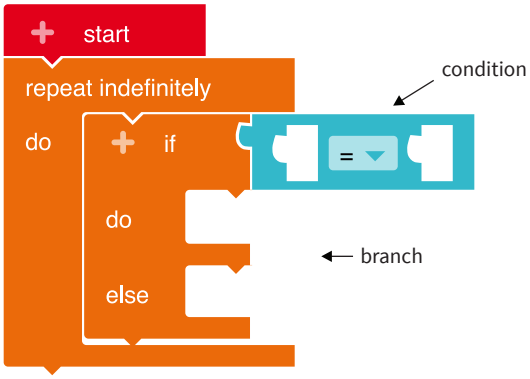
Steps to create the program for an automatic bicycle rear light

1 So that the *Calliope mini* can react to a change in the incidence of light at any time, the light intensity must be measured continuously: For this you need an infinite **loop.** Any **blocks** you put in the infinite loop are repeated over and over. Select the "Repeat/do" block from the category **Control** and add it to the "Start" block.

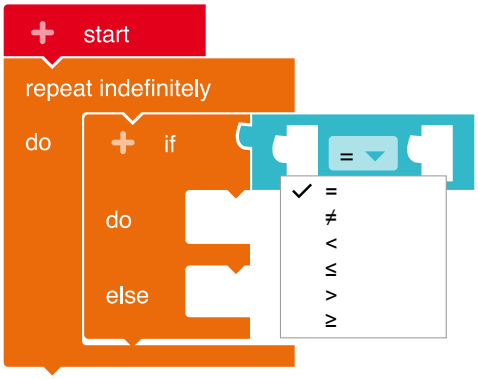
2 The light intensity measured should be divided into two areas: “dark” and “light”:
For this you need a **branch**.
Select the “if/do/else” block from the category **Control** and insert it into the **loop**.
This branch, based on a **condition**, is used to differentiate between two alternative **program** processes.



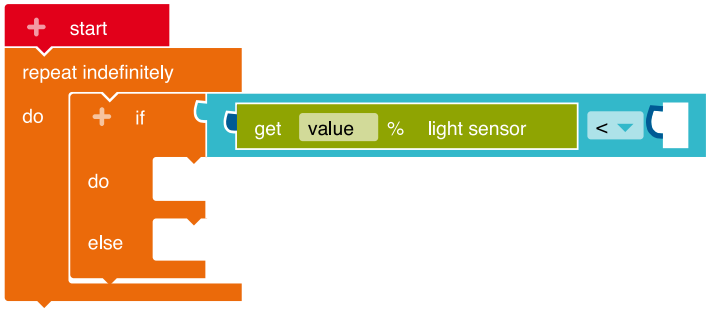
3 The threshold value must be checked in order to distinguish between “light” and “dark”:
For this you need a **comparison** between two numbers.
Select from the category **Logic** the block  . Add it as a **condition** to the **branch**.



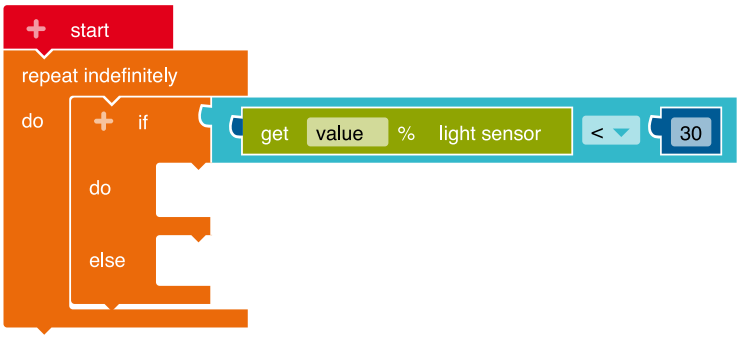
4 The light should be turned on as soon as it is dark enough:
For this, the following **condition** must be fulfilled: The measured value of the ambient light falls below a specified threshold value. The “=” sign (equals) is unsuitable for this and the “<” sign (less than) must be selected.
To do this, click on the “=” symbol and select “<”.



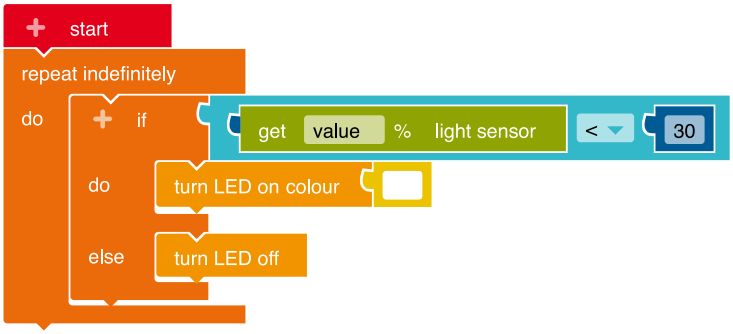
5 The intensity of the incidental light must be measured by the *Calliope mini* and provided as a numerical value for **comparison**:
Select the block “Enter value for ambient light” from the category **Sensors** and insert it into the left-hand space of the block “is smaller than”.
The *Calliope mini* determines the light intensity as a number between 0 (no light) and 255 (very bright).



6 The comparison value for the light intensity determined must be specified:
Select block “0” (zero) from the category **Mathematics** and insert it into the right-hand space of the block “is less than”. Then click on the field with the zero and enter a number between 0 and 255. You can determine the desired value by trial and error. A good start value is 30. This defines the threshold value, which indicates the light intensity from which the area around the *Calliope mini* is considered “dark”.
The **condition** is now complete.
It applies when the level of ambient light is less than the set value.



7 The **RGB-LED** should be turned on if the **condition** applies that means it is “dark”:
Select the block “Turn LED on colour” from the category **Action** and insert it into the **branch** at the gap marked “do”. In the event that the condition does not apply, i.e. it is “bright”, the RGB LED should be turned off.
Select the block “Turn LED off” from the category **Action** and insert it into the branch at the gap marked “else”.



8 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

In daylight or artificial light, the RGB LED should be turned off after the program has started. Place the *Calliope mini* in the shade, for example by shading it with your hands, placing it in a shoe box with a lid, or holding it under your desk. The RGB LED should now turn on. If it does not, increase the threshold a little (see step 6) so that the environment is classified as “dark” sooner.

Application reference

Light sensors are used in environments in which it is necessary to create additional brightness when the natural lighting is too low. These include, for example, the automatic dipped beam in vehicles, an automatic flash in a camera, the regulation of the display brightness on smartphones or street lights that turn on in the evening and turn off in the morning.

Expansion possibilities

Instead of just turning on the RGB-LED, the students can also run self-created image sequences on the LED screen when the *Calliope mini* is in the dark. A short pause (“wait ms” block) is inserted between the images.

The *Calliope mini* as a mini piano

The exercise

The *Calliope mini* is first programmed to output individual tones, which are output via the tone generator. The contact between the buttons created by the touch of the hands with the Touch-Pins is the trigger for the sound output.
The first program already contains all the basic structures that are necessary to complete the mini piano.

Subject matter

Subject teaching

With the mini piano, students in the field of *Technology*can learn that their bodies conduct electricity. With their fingers they create a connection between two contact surfaces on the *Calliope mini*. Conductive connections are indicated with different tones. In this way a mini instrument with four different tones is achieved.

Position in the syllabus

Content-related skills

The students explore selected body parts in the field of *Nature and life*. By creating a connection between two *Calliope mini* contact surfaces with your fingers, you will learn that your body conducts electrical current.

Process-related skills

In the field *World*, the students explore and understand methods of exploring and gaining knowledge (e.g. observing, experimenting, dealing systematically) and applying them.

Requirements

Programming focus:

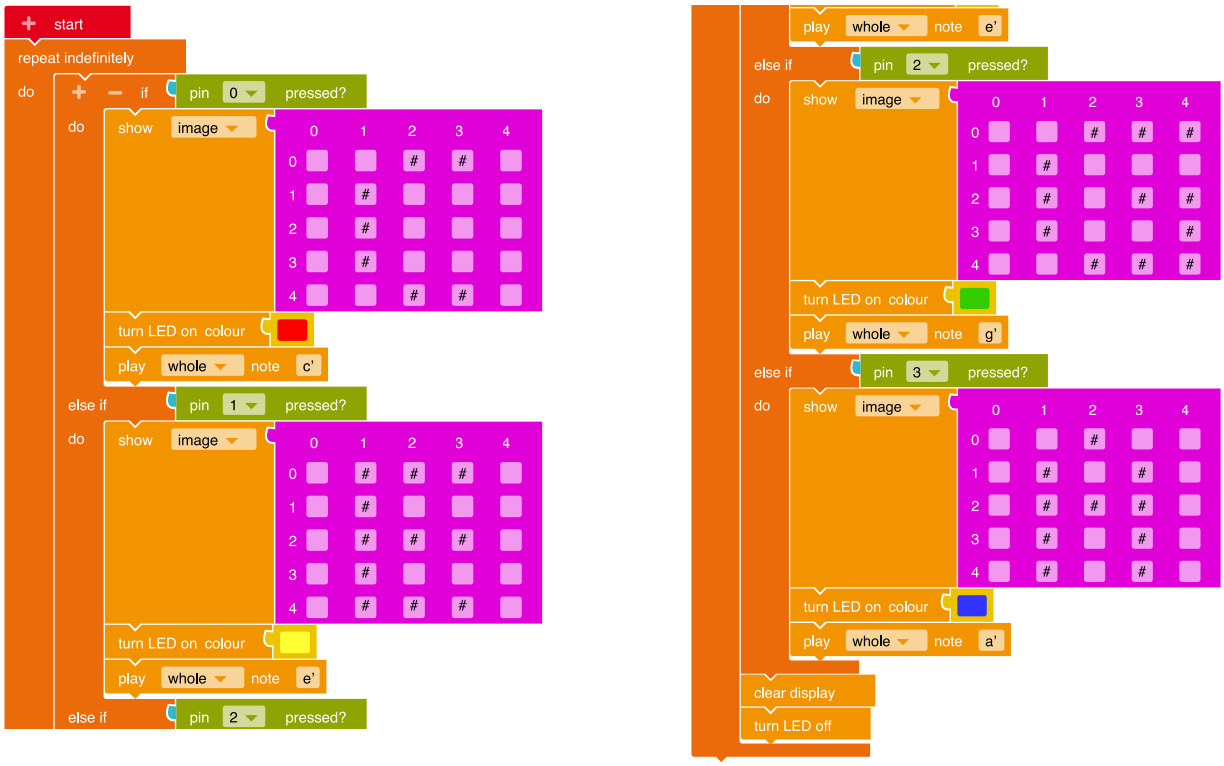
- Input by means of Touch-Pins, output via speaker and LED screen
- Structures: Infinite loop, branch, command
- *NEPO*® categories used: Control, action and sensors

Programming difficulty:

- Beginner level
- *NEPO*® Level: Beginner

The code

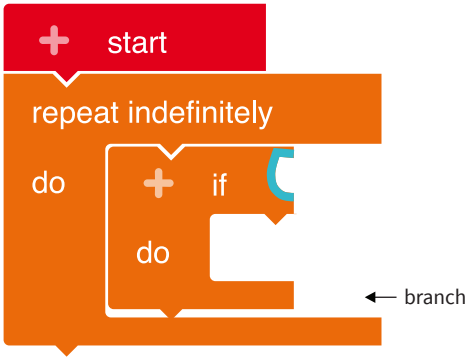
This is what the code of the finished program looks like. It is defined step by step below.



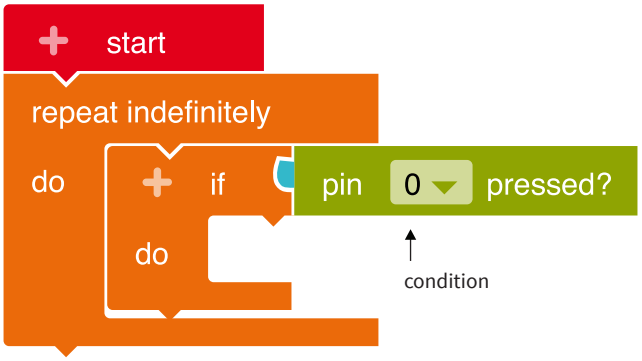
Steps to create the program for a single tone

1 You need an infinite loop so that the *Calliope mini* can react to any number of consecutive Touch-Pin contacts: Select the "Repeat indefinitely/do" block from the category **Control** and add it to the "Start" block.

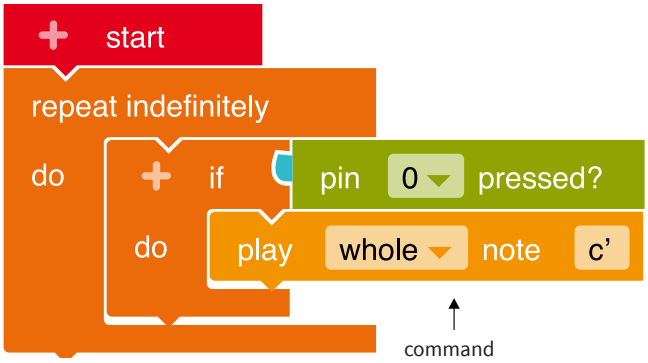
2 The Calliope should generate a tone if a Touch-Pin is touched. For this you need a **branch**.
In the category **Control**, select the "if/do" block and insert it into the loop.



3 In order to decide whether a Touch-Pin has been touched or not, the *Calliope mini* must determine its status and provide it as a **truth value**:
Select the "Pin 0 pressed?" block from the category **Sensors** and add it to the branch as a **condition** (blue area).



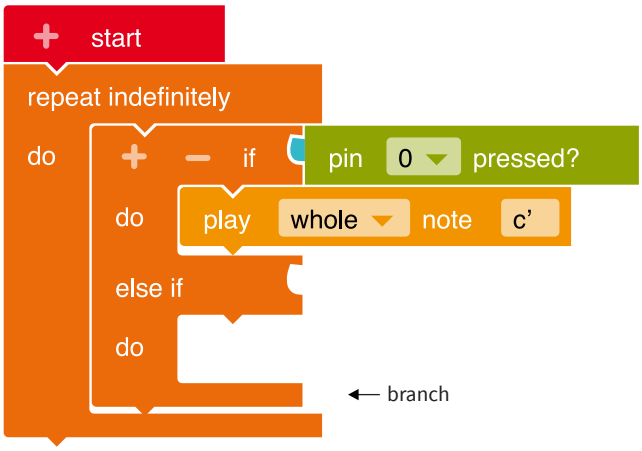
4 A tone should be played if the condition applies meaning that the Touch-Pin is touched:
Select the "Play whole note c" block from the category **Action** and insert it into the gap in the branch marked "do". This **command** has the *Calliope mini* generate the specified tone.



5 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

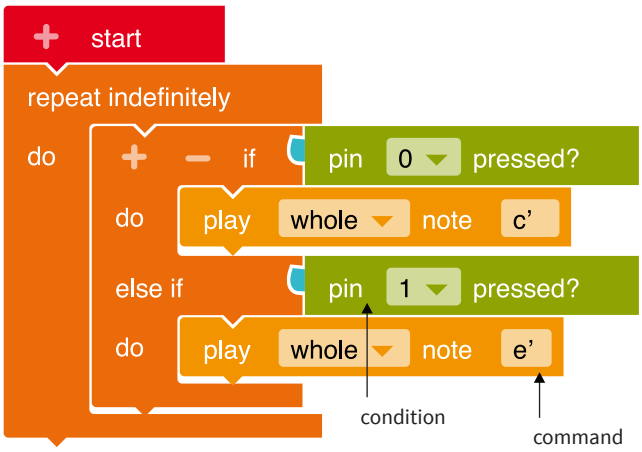
Steps to create the program for multiple tones

1. Expand the program to query the other **Touch-Pins** so that additional tones can be played:
To do this, click on the "+" next to the "if". Another **branch** appears. The new branch is executed if Pin 0 *is not* connected to the Minus-Pin. This gives the *Calliope mini* the opportunity to check another Touch-Pin.



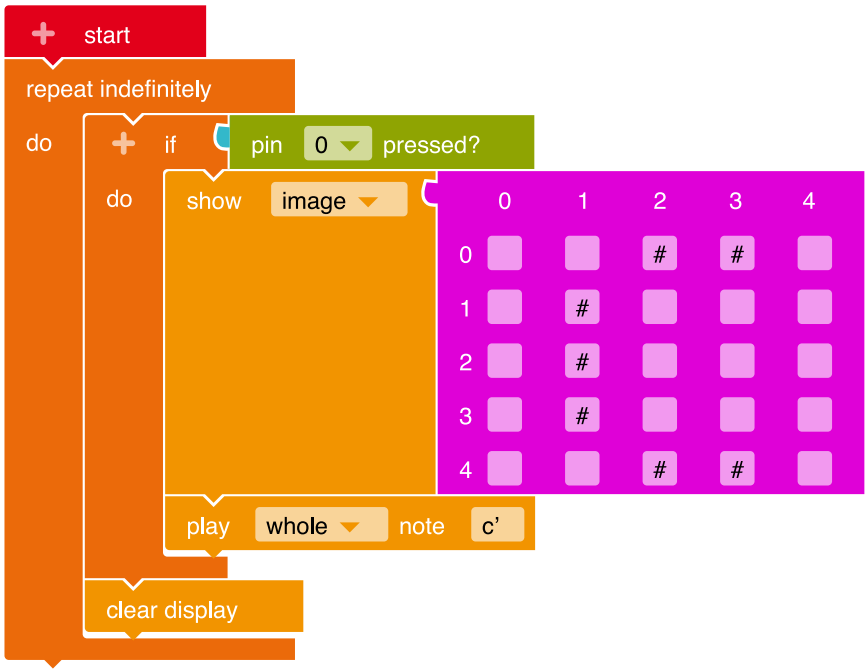
2. Set which note the *Calliope mini* should play when Touch-Pin 1 is touched:
From the category **Sensors** add the **condition** for checking the Touch-Pin 1. Then from the category **Action** insert another **command** for sound output and select the desired sound. An example was selected in the figure on the right.

3. Expand the program so that all four Touch-Pins can be checked and four different tones can be output.



4. Expand the program so that the name of the note that is currently being played is displayed on the **LED screen**. See the illustration below for an example of a single tone.

Select from the category **Action** the block "Show image" and insert it into the gap marked with "do" of the **branch**. Select the fields in the 5 x 5 grid with a click so that the note you are currently playing is displayed.



5. Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

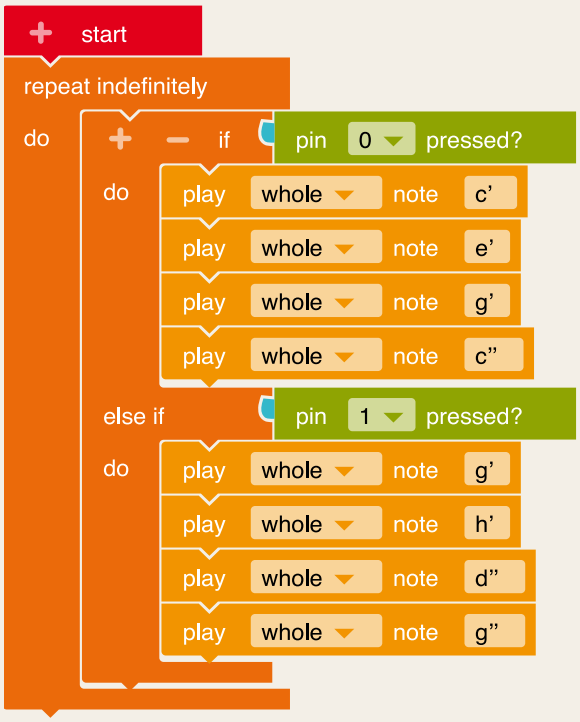
When connecting the Touch-Pins, one finger must rest on the Minus-Pin. To do this, hold the *Calliope mini* in your left hand and enclose the Minus-Pin with your thumb and index finger. Touch Pins 0 to 3 with one finger of your right hand to start the sound output.

Application reference

Among other things, the topic connects the subject lesson in a very clear way with the subject of music by showing the basic functionality of electrical instruments. Keyboards and electric pianos work on the same basic principle as the program: A tone is emitted when a **button** is pressed, meaning that the circuit is closed.

Expansion possibilities

As an expansion of the program, different chords can be programmed for each pin. When a contact is made, a short tone sequence is played instead of a single tone. For example, a C major chord can be played when you touch Pin 0 and a G major chord when you touch Pin 1. This is how a song can be accompanied.



The *Calliope mini* as a metronome

The exercise

The *Calliope mini* offers the possibility of timing the sequence of a program. The accuracy is in the millisecond range. This creates a metronome that can be used, for example, to display different heartbeat frequencies.

The program lets the *Calliope mini* generate outputs periodically, for example flashes of light or tones.

Subject matter

Subject teaching

The students learn to use the time measurement in the *Calliope mini* to build their own electronic metronome. A strict time sequence is required for this, which is implemented using the built-in waiting time blocks.

Position in the syllabus

Content-related skills

In the field of *Nature and life* , the students can describe selected body parts, for example by comparing the metronome with their heartbeat.

Process-related skills

In the field of *Exploring and understanding the world*, the students can compare, classify and relate experiences to different contexts (e.g. with regard to sense of time and time awareness).

Requirements

Programming focus:

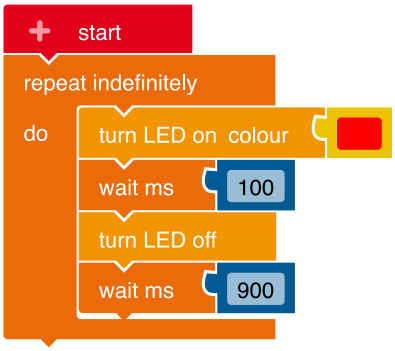
- Control of the program sequence by means of waiting times
- Structures: Infinite loop, command
- *NEPO®* categories used: Control and action

Programming difficulty:

- Beginner level
- *NEPO®* Level: Beginner

The code

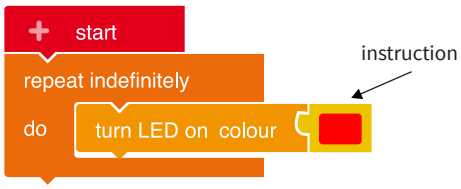
This is what the code of the finished program looks like. It is defined step by step below.



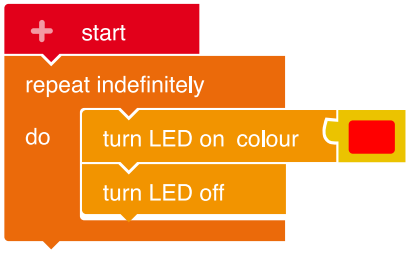
Steps to create the metronome program

1. The clock pulses should be output without interruption: For this you need an infinite **loop**.
Select from the category **Control** the **block** "Repeat indefinitely/do" and add it to the "Start" block.

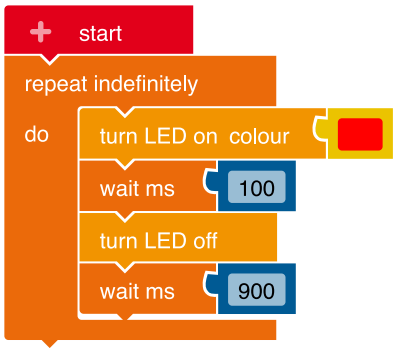
2. The light flashes are created by the time-controlled turning on and off of the **RGB LED**:
To do this, you need an **instruction** to turn on the RGB LED.
To do so, select from the category **Action** the block "Turn LED on colour" off and insert it into the loop.



3. The instructions for turning off the RGB-LED can also be found in the category **Action**:
Add the block "Turn LED off" to the program.



4. In order for the turning on and off to be visible, **Waiting** times must be determined which determine how long the RGB LED lights up or is turned off: To do this, select the "Wait ms" block from the category **Control**, which stops the program for the specified number of milliseconds and then continues it again.
The RGB LED is turned on and turned off again after a tenth of a second (100 milliseconds). For the rest (900 milliseconds) up to the full second (1000 milliseconds) it is dark, that means it flashes once per second, which corresponds to a cycle of 60 beats per minute.
If a faster cycle is required, the turn-off time must be reduced. The cycle slows down accordingly if the turn-off time is extended.



5. Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Expansion possibilities

The tempo of the metronome can be changed by pressing a button.

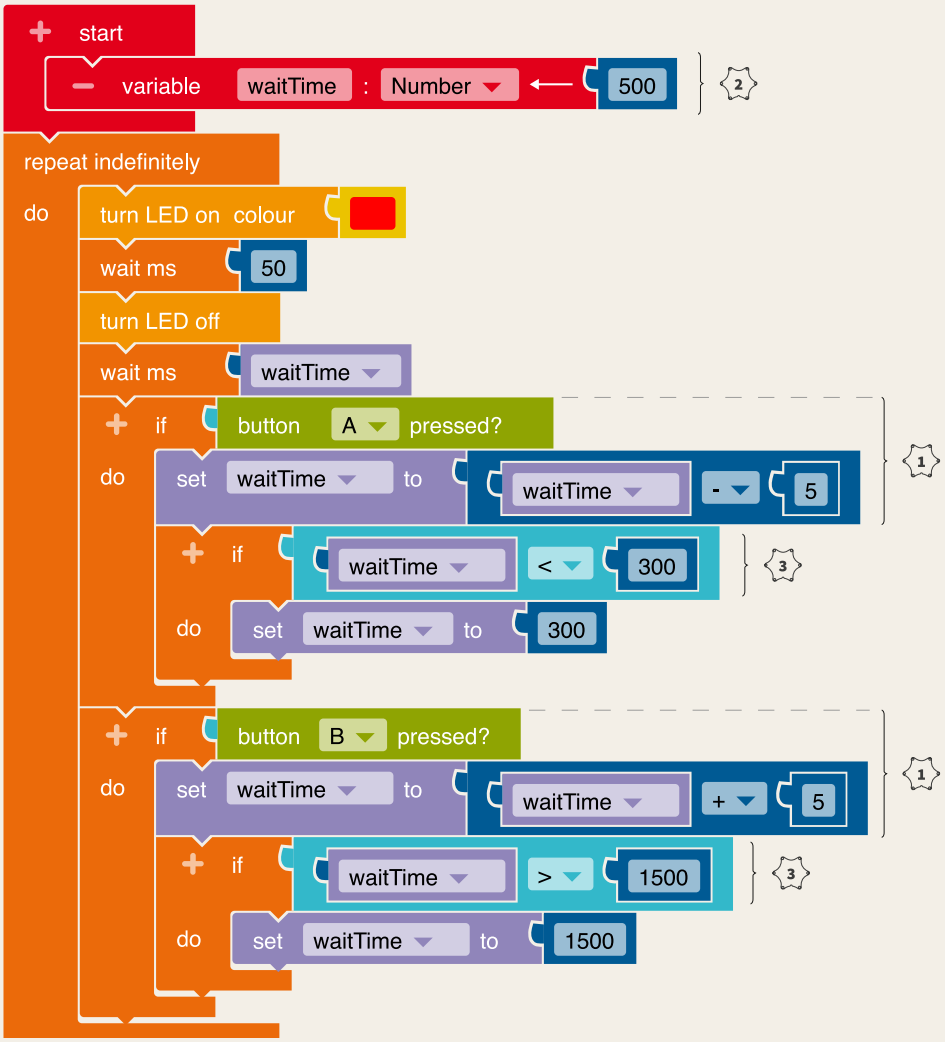
(1) Each time the **Button** A is pressed, the metronome should become 5 (ms) milliseconds faster: The turn-off time of the RGB-LED is reduced by 5 ms. Each time you press Button B, however, the metronome slows down by 5 ms.
For these changes in the clock frequency, the current value of which is stored in the **variable** *Waiting time*, the **waiting time** is entered accordingly.

(2) Create variable: Click on the "+"-symbol next to the "Start" block.

(3) To prevent the metronome from flashing too fast or too slowly, the waiting time is limited to a lower limit of 300 ms and an upper limit of 1500 ms.

Implementation scenarios in class

The students compare the clock rate of the *Calliope mini* with their heart rate by feeling their pulse and at the same time watching the *Calliope mini*. By adjusting the turn-off time (lower waiting time in the program) you try to adapt the clock generated by the *Calliope mini* to your pulse.
Areas of application: Metronomes can be used as timing instruments by controlling counters. Musicians also use these devices for orientation when making music together or for studio recordings. Flashing lights to warn vehicles or construction sites also use metronomes.



The *Calliope mini* as a stopwatch and countdown counter

The exercise

With this program the *Calliope mini* can be used as a stopwatch. To start, the Button A is pressed once. At the same time as the stopwatch starts, the RGB LED lights up green. Pressing Button B stops the clock and the time recorded is displayed in seconds on the LED screen. At the same time, the colour of the RGB LED changes from green to red. The pressing of the button can be supported acoustically by a short tone.

A variant of the stopwatch is the countdown of a period of time with a countdown counter (alarm clock). The countdown starts by pressing "A". For every second an animation is shown on the LED screen and the RGB LED lights up green. After a time previously set in the program, for example one minute, the RGB-LED turns to red, a final image appears and an alarm tone sounds ten times in a row.

Subject matter

General knowledge

The students encounter the phenomenon of time in a variety of ways. They can orientate themselves in manageable periods of time, differentiate between time concepts and time structures and apply them. The *Calliope mini* is a time measurement instrument that can be used in many ways.

Position in the syllabus

Content-related skills

In the field of *Time and change*, the students can use linear concepts of time and instruments (clock, ...) as well as understand time as a finite and infinite phenomenon and relate experienced and measured time by using the stopwatch and countdown timer as a timer and for timing control.

Process-related skills

In the field of *Exploring and understanding the world*, the students can understand time as a finite and infinite phenomenon, relate experienced and measured time to each other, and compare and organise corresponding experiences and relate them to different contexts (for example sense of time and time awareness).

Requirements

Programming focus:

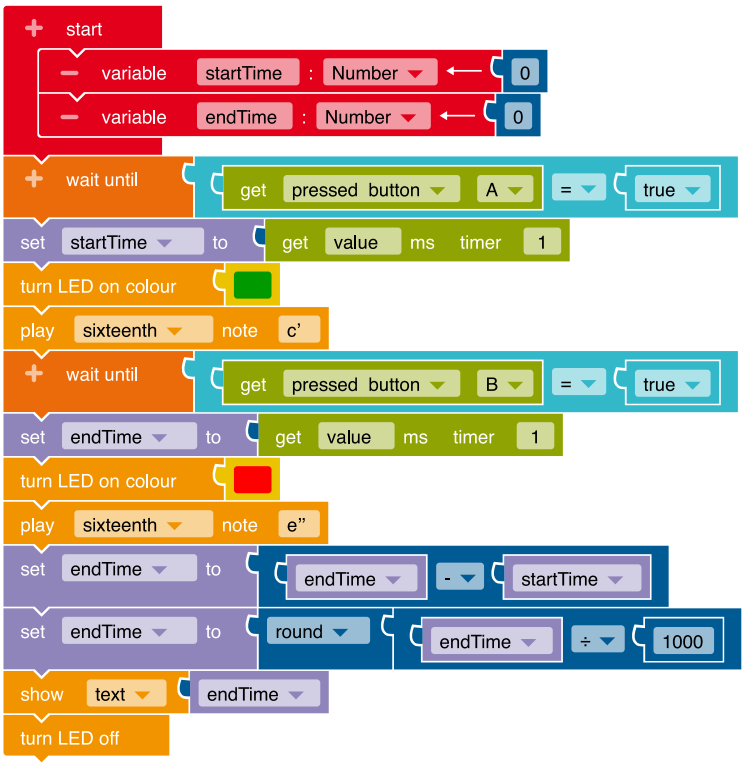
- Input via buttons, output via LED screen, RGB LED and loudspeaker
- Structures: conditional waiting, counting loop
- calculate with variables
- Query timer
- Use functions (countdown counter)
- *NEPO*® categories used: Action, Sensors, controls, logic, mathematics, variables, functions

Programming difficulty:

- Beginner level
- *NEPO*® Level: Beginner/Expert

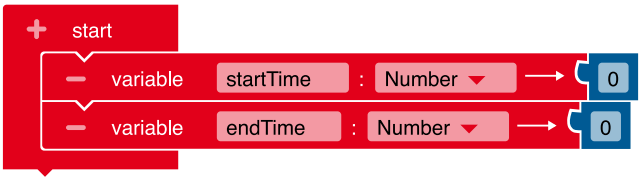
The code for the stopwatch

This is what the code of the finished program looks like. It is defined step by step below.

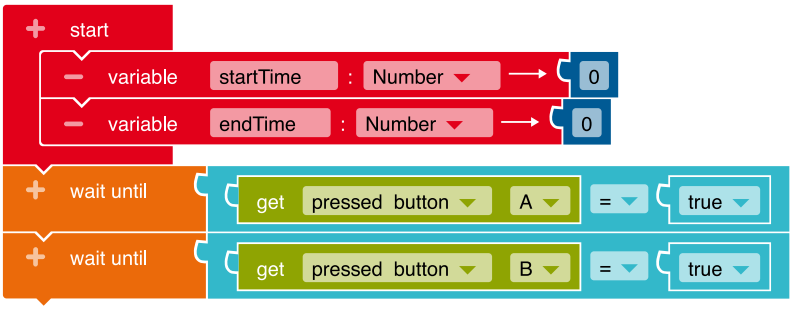


Steps to create the program for a stopwatch

1 So that the start time and the end time of the stopwatch can be saved, two **variables** must be created:
To create a new variable, click on the "+" to the left of "Start".
A new block appears. Define the names (*startTime, endTime*) and the **data type** (number) of the variables. Both variables initially have the value zero.

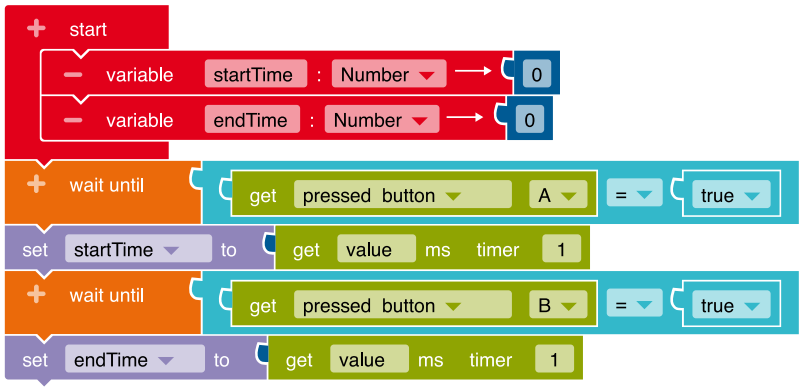


2 Two **instructions** are required for button control of the stopwatch:
Select the "Wait until" block from the category **Control** and add it twice.
This block already contains the query as to whether Button A has been pressed.
In the second block, change Button "A" to button "B".



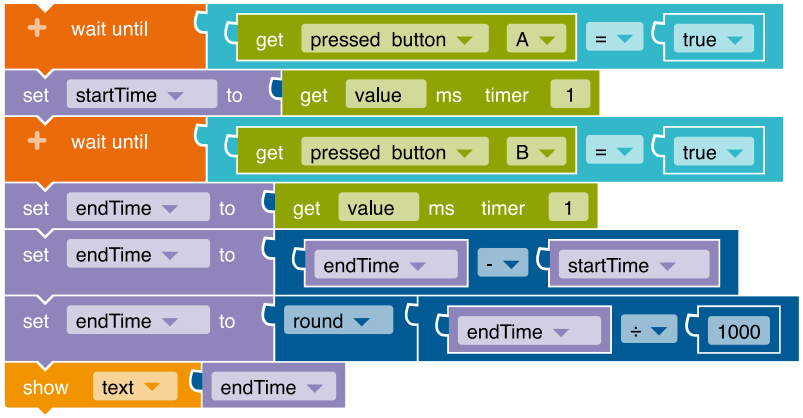
3 Following the button queries, two **instructions** for saving the time values are added:
Select the block "Write" from the category **Variables** and insert it under the **Wait** blocks.

Select the **variable** *startTime* in the first block and the **variable** *endTime* in the second.
From the category **Sensors** dock the block "get value "timer 1" in ms" to the new blocks.



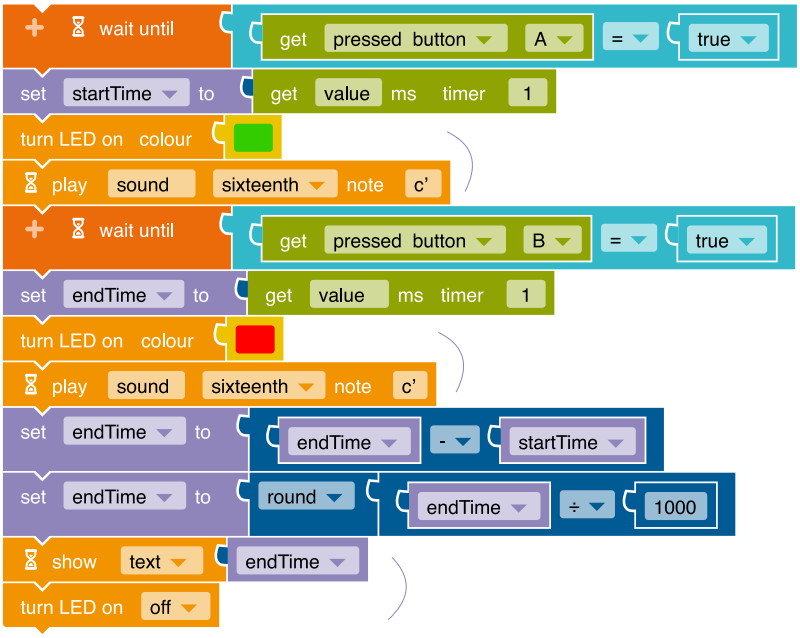
4 Now the elapsed time can be calculated in the **variable** "write *endTime*":
Select the arithmetic block from the category **Mathematics**, change it to subtraction and insert the variables *endTime* and *startTime*.
In the second step, whole seconds are calculated by dividing by 1000 and rounding off the thousandths of a second (switch to expert mode for this block).

The time is displayed on the LED screen:
To do this, select the block "Show text" from the category **Action** and replace the text "Hello" with the variable *endTime*. To do this, select from the category **Variables** the block *endTime*.



5 Additional **outputs** of the *Calliope mini* complement the stopwatch program:
Select the blocks "Turn LED on colour" and "Play note" from the category **Action**.
At the end the RGB LED is turned off again.

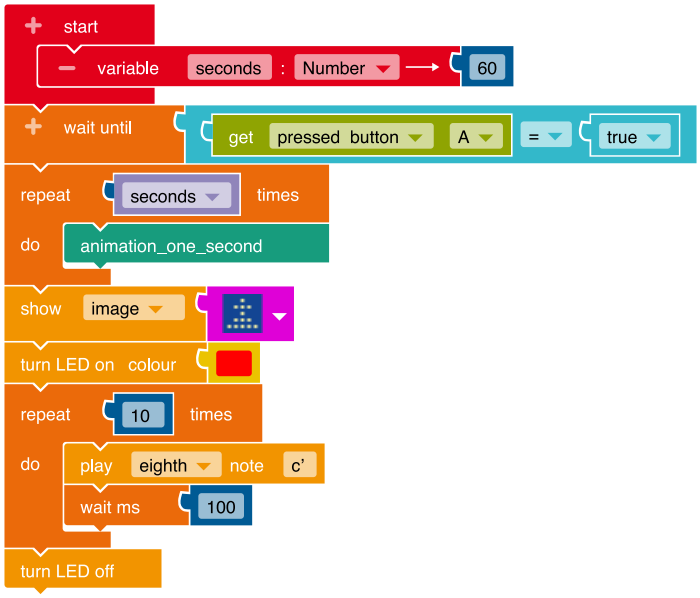
6 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).



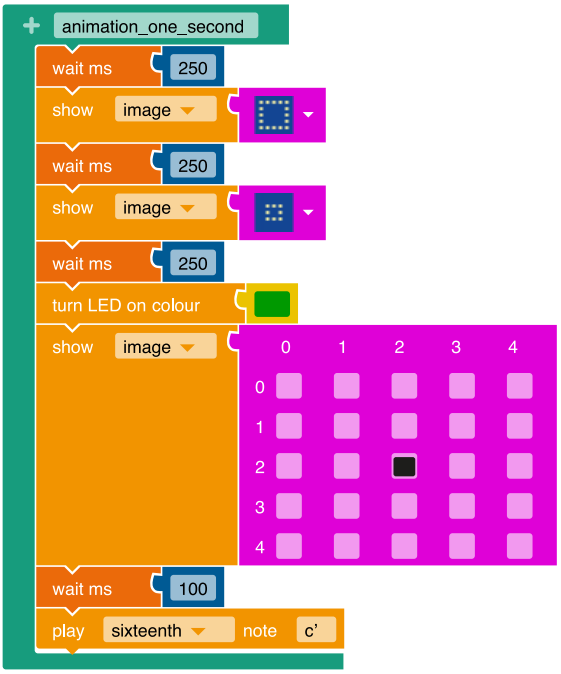
The code for the countdown timer:

This is what the code of the finished program looks like. It is defined step by step below.

Hauptprogramm



Funktion

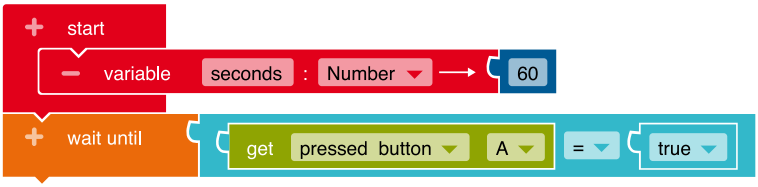


Steps to create the program for a countdown timer (alarm clock)

1 First a **variable** is required for the time span of the countdown to be saved:

To create a new variable, click on the "+" to the left of "Start" block. A new block appears. Define the name (*seconds*) and the **data type** (number) of the variable. The selected start value "60" indicates the seconds of the countdown.

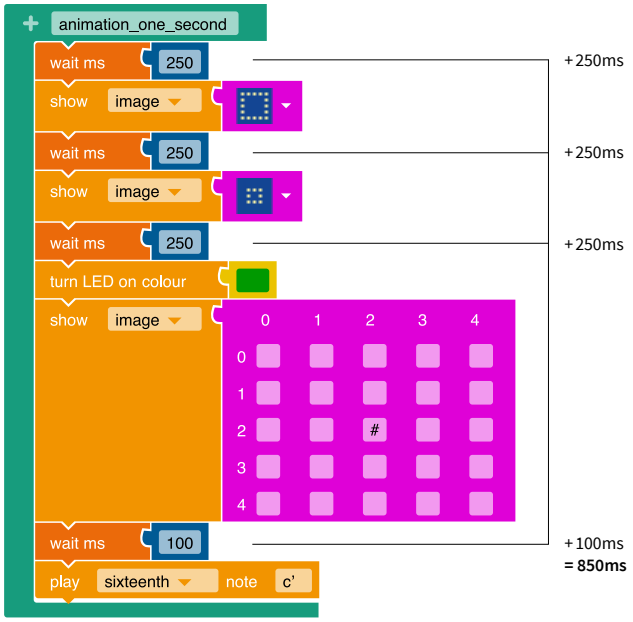
2 To start the countdown while the program is running, pressing Button A must be registered: Select the "Wait until" block from the category **Control** and add it to the "Start" block. This block already contains the query as to whether Button A has been pressed.



3 The animation that runs within one second is programmed outside of the main program in a **function**:
Select the "doSomething" block from the category **Functions** and place it in a free window area next to the main program and change the name to *animation_one_second*.

Within the **function** add the **action** blocks for displaying the images on the LED screen, as well as the waiting blocks from the category **Control** as shown on the right.

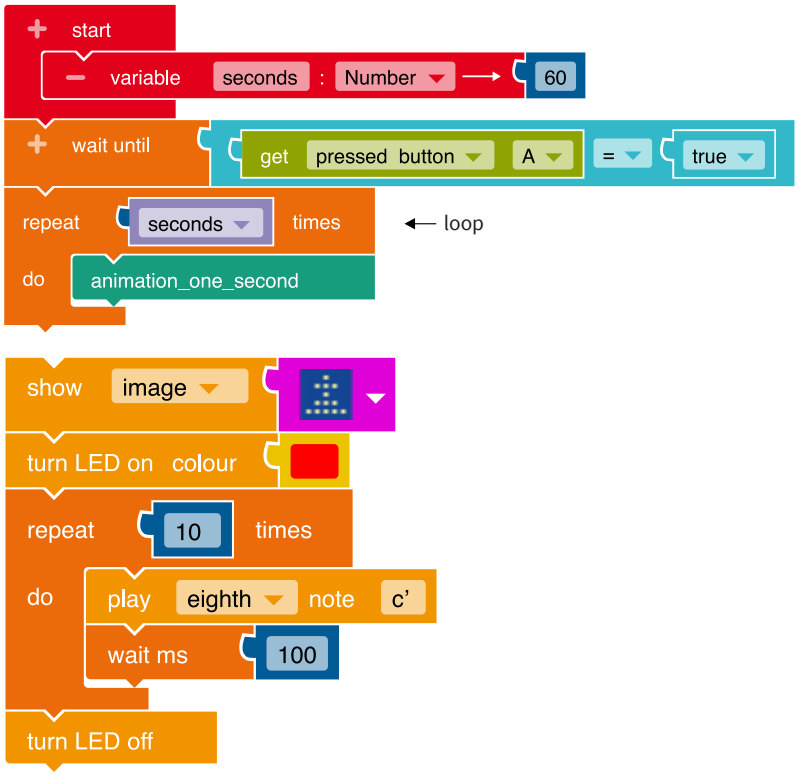
The total of the waiting time blocks is only 850 milliseconds. The remaining time up to the full second is consumed by the animation.



4 The seconds animation programmed within the function must be repeated 60 times according to the start value:
Select from the category **Control › Loops** the block "Repeat 10 times" and replace the value 10 with the **variable** *seconds*. In this counting **loop** insert the previously created block *animation_one_second* from the category **Functions**.

5 When the loop ends, the countdown has ended. This should be indicated by the following outputs:
Select from the category **Action** the block "Show image" with a suitable representation and for example further **commands** for the RGB-LED or a sound output.

6 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).



Notes and information

Implementation scenarios in class

- Time estimates, for example for the duration of a task being processed
- Time measurements in experiments
- Time measurement in competitions
- Relationship between distance and time

Expansion possibilities

- The stopwatch can be reprogrammed so that measurements can be taken in smaller time units. For tenths of a second the divisor is changed from 1000 to 100.
- Additional output of tones or light signals for every minute / second (e.g. for time measurements in games / sports).

Morse code with the *Calliope mini*

The exercise

In this exercise, the *Calliope mini* is programmed so that it can be used as a Morse code device. To display Morse code, all that is required is the output of a point and a horizontal line on the LED screen. The Morse code is audibly supported by additional programming of tones of different lengths and the self-coded mini Morse code also works when the sender and recipient cannot see each other.

Subject matter

English

The students have to deal with foreign scripts and systems of symbols. They will learn Morse code, discuss how it can be represented and the advantages of programming Morse code on the *Calliope mini* (e.g. simultaneous visual and auditory output, functionality even in the dark, etc.).

Position in the syllabus

Content-related skills

In the field of *Language and use of language*, the students can express themselves on issues in a structured manner, also using digital communication media, by dealing with foreign scripts and symbol systems.

Process-related skills

In the field of *Speaking and listening* the students can use media as a means of everyday communication.

Requirements

Programming focus:

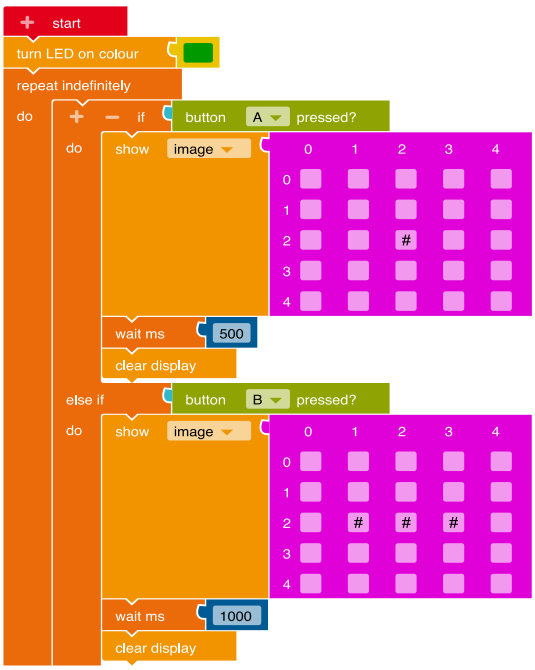
- Input via buttons and Touch-Pins, output via LED screen, loudspeaker and RGB LED
- Structures: conditional statement, infinite loop, formulating conditions
- *NEPO*® categories used: Action, sensors, control

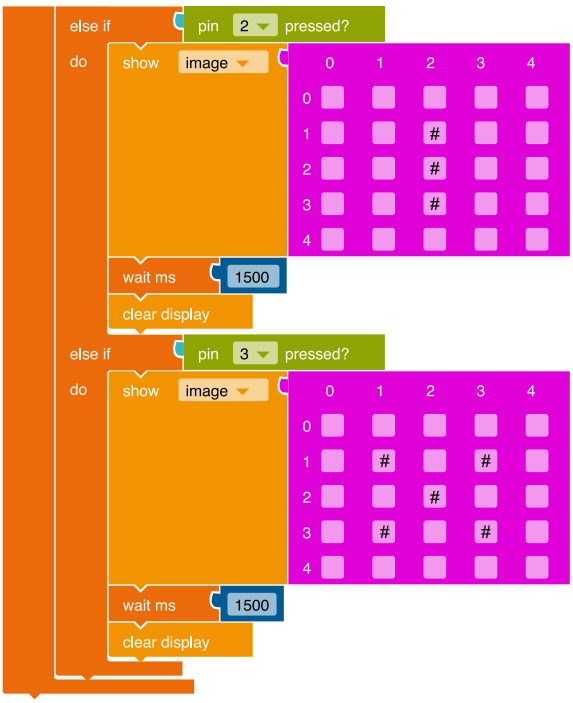
Programming difficulty:

- Beginner level
- *NEPO*® Level: Beginner

The code

This is what the code of the finished program looks like. It is defined step by step below.



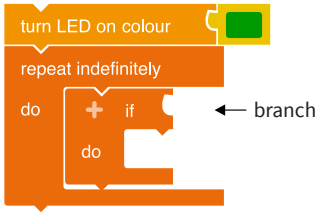


Steps to create the program for the output of Morse code

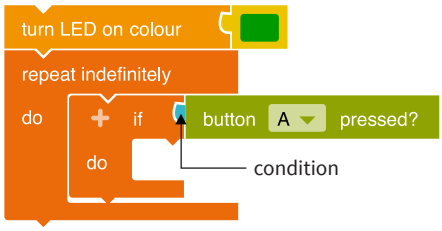
1. So that the user can see that the Morse code is in operation, the **RGB-LED** is switched on immediately when the program starts:
Select the block “Turn LED on colour” from the category **Action** and add it to the “Start” block.

2. An infinite loop is required so that Morse code can be displayed infinitely:
Select from the category **Control** the block “repeat indefinitely/do” and add it.

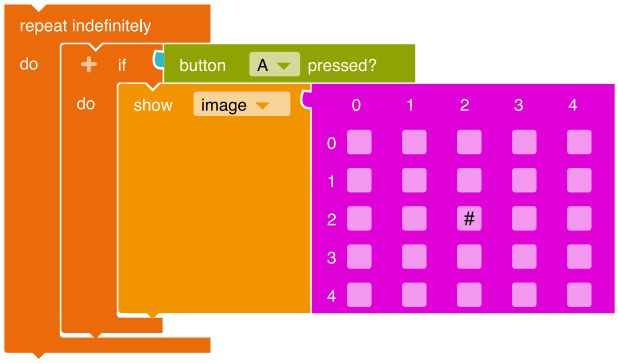
3. When the **Button A** is pressed (*if*), a Morse point should be displayed (*do*):
To establish this condition, you need a **branch**.
Select the block “if/do” from the category **Control** and put it in the loop.



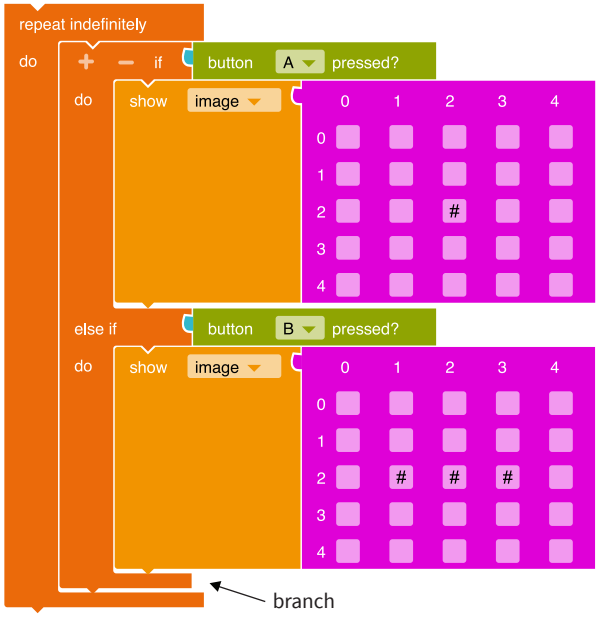
4. The desired input option (here via the sensor “Button A”) is selected:
Take from the category **Sensors** the block “Button A pressed?” and append it to the branch as a **condition** (blue area).



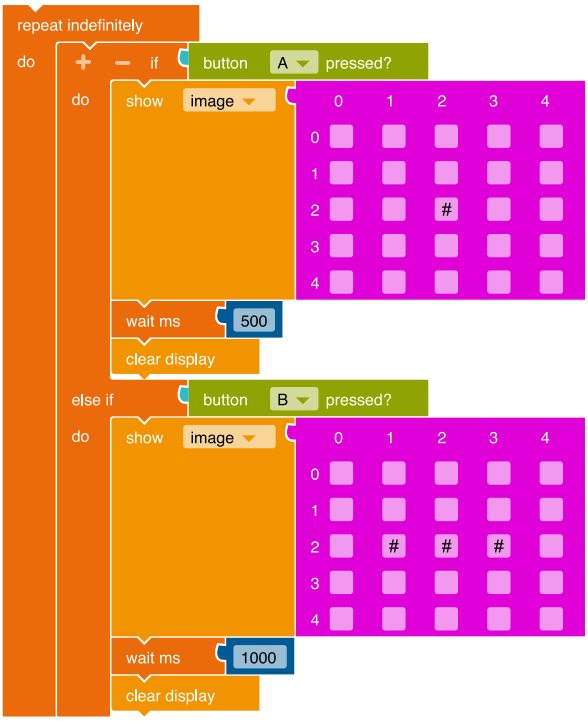
5. The Morse point is shown on the LED screen:
Select the block “Show image” from the category **Action** and mark the box in the middle of the 5 x 5 grid to display the Morse point.



6. When Button B is pressed (*if*), a Morse line should be displayed on the LED screen (*do*):
For this condition, you need another **branch**.
To do this, click on the “+” next to the “if” in the branch.
Select from the category **Sensors** the block “Button B pressed?” and insert it as a **condition** (blue area) in the new branch.
Select the block “Show image” from the category **Action** and mark three points in the horizontal of the 5 x 5 grid to display the Morse line.



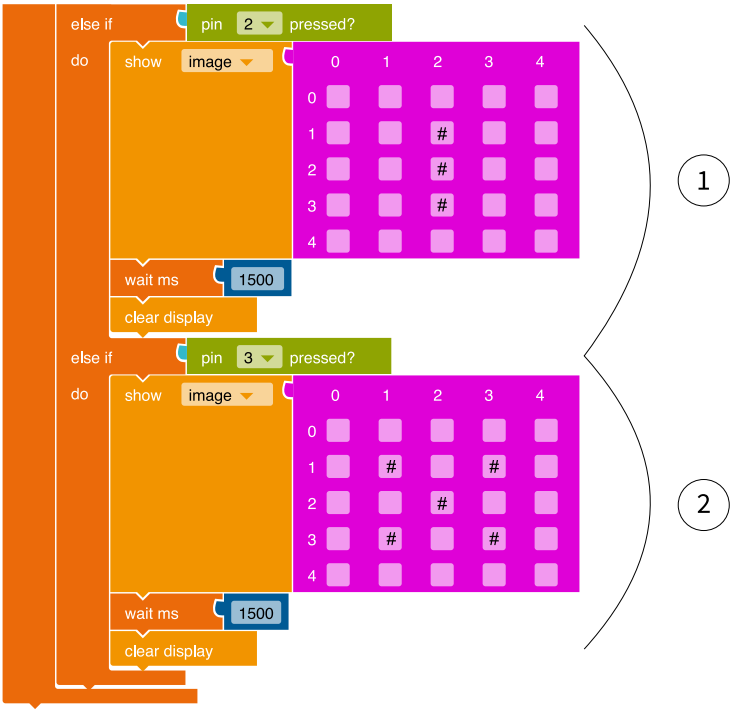
7 In order for the Morse characters to be legible, they must be separated from one another. For this purpose, a **pause** is inserted after each output and then the screen content is cleared:
Select the “Wait ms” block from the category **Control** and define the length of the pause here (1000 ms = 1 second).
Now delete the screen using the “Clear display” block from the category **Action**.



8 In order for the recipient to be able to differentiate between letters and words clearly, the code needs to be expanded by further branches:

(1) Delimitation letter:
To do this, select the “Pin 2 pressed?” block from the category **Sensors** and add it to another branch. Use the “Show image” block to create a vertical line from the category **Action** to separate individual letters.

(2) Delimitation word:
Create another branch and add from the category **Sensors** the block “Pin 3 pressed?” With the block “Show image” from the category **Action** you create a cross to delimit individual words.



9 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

This code works well for working in pairs. As an extension, the students can program an automatic Morse word of their choice.

Background knowledge

The artist and inventor Samuel Morse developed the first telegraph from 1837, which enabled fast communication over very long distances. Instead of words, the telegraph could only send electrical impulses. Fixed combinations of short and long power surges represent the respective letters.

Notes that connect subjects

The topic *Morse code* can be well located in the syllabus of the subject teaching:
The students can use different types and methods of communication, describe and reflect on selected inventions, their development and the impact on the living environment, also with a view to the future.

Expansion possibilities

Supporting sound output:
As an extension of the program, the Morse code can be supported audibly by a sound output. To do this, select the block “Play whole note C” from the category **Action** and insert it under the block “Show image”. Choose a suitable pitch and length.

Note: The block “Wait until” is no longer necessary, as the length of the tone also determines the length of the display of the Morse code on the LED screen.

Transmit Morse code with the *Calliope mini*:
The category **Messages** offers the possibility to let several *Calliope mini* microcontrollers communicate with each other. In this way a Morse code can be sent from *Calliope mini* to *Calliope mini* or to different boards at the same time.

Generate image impulses and stimulus words with the *Calliope mini*

The exercise

The *Calliope mini* is programmed as an idea generator, which should automatically generate suggestions for something to write. For this purpose, one of three or more images is randomly selected and output at the push of a button. In addition, the description of the picture appears as a single word shortly thereafter. The images and words available in the *Calliope mini* can be designed or freely chosen.

Subject matter

English

The students develop ideas for a short story and plan it based on an image displayed on the 5 x 5 LED screen. At the same time, the clarity and meaning of pictograms in different representations can be discussed.

Position in the syllabus

Content-related skills

In the field of *dealing with text and other media*, the students can plan and write their own text based on suggestions by using the image/word presented as a basis and writing a stimulus word story.

Process-related skills In the field of *Writing*, the students can develop, plan and write a writing idea and pay attention to the logical sequence and, depending on the reason for writing, write appropriately for the addressee and the function.

Requirements

Programming focus:

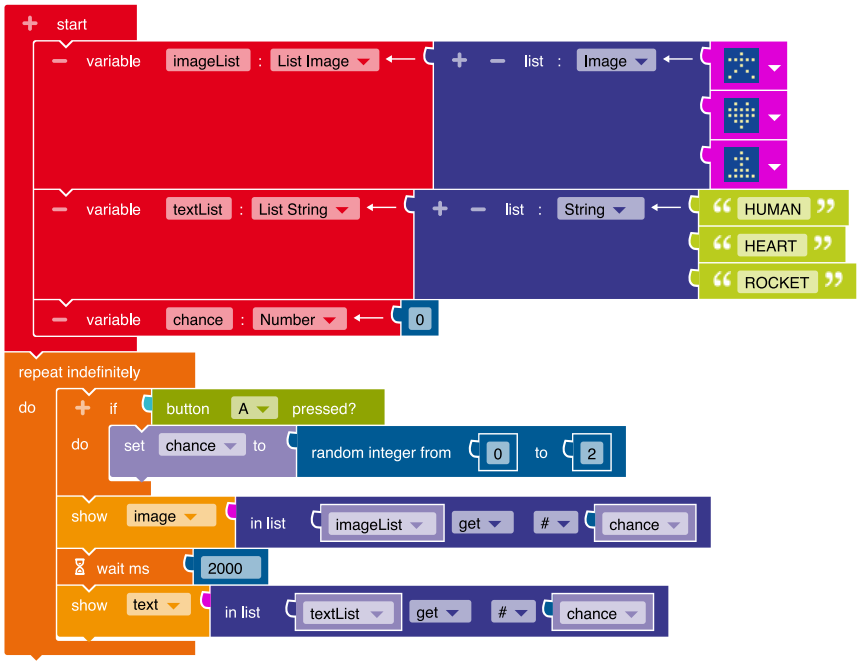
- Input via buttons, output via LED screen
- random selection of an element from a list
- Structures: Infinite loop, variable, list, random, branch, Command used *NEPO*®-categories: Control, action, list, variables, mathematics, images, sensors

Programming difficulty:

- Advanced level
- *NEPO*® Level: Expert

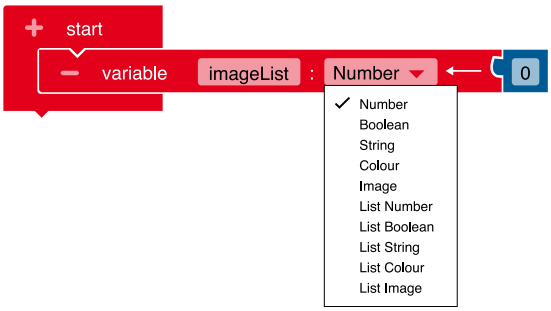
The code

This is what the code of the finished program looks like. It is defined step by step below.



Steps to create the program for the image impulses and stimulus words

1. The images and words available should be saved in **lists**: First, create a list that includes the images available. To do this, create a new **variable** by clicking on the "+" next to "Start".
Change the name of the variable in the *imageList* and select in the **drop-down menu** "List Image" as the **data type**.



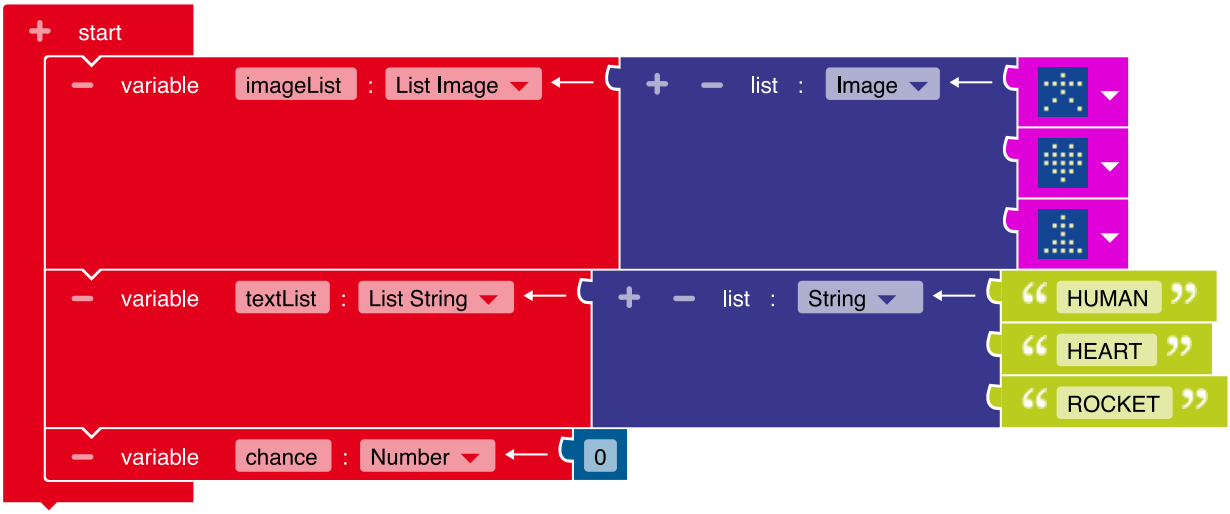
2. It starts with the images:
Select the images that the *Calliope mini* should later display on the **LED screen**.

You can start with the images provided in *NEPO®* or create your own images. The required blocks can be found in the category **Images**.

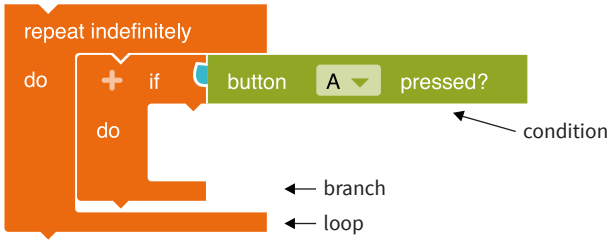
3. The words that are to be displayed for the images must also be stored in a list:
Create two more variables. Name the second variable *textList*. Select "List **String**" as the data type. Then you can enter the words that match the images in the green text blocks, making sure that the words are in the same order as the images. In the example, the stick figure is in the first **position** in the image list and the word "HUMAN" is in the first position in the text list.

The images and texts should be selected randomly by the *Calliope mini*:

Create a third variable that is used to hold the **random** number generated. Name this one *chance* (Names like "random" are already used in the systemcode and the editor therefore adds numbers to the variable name).

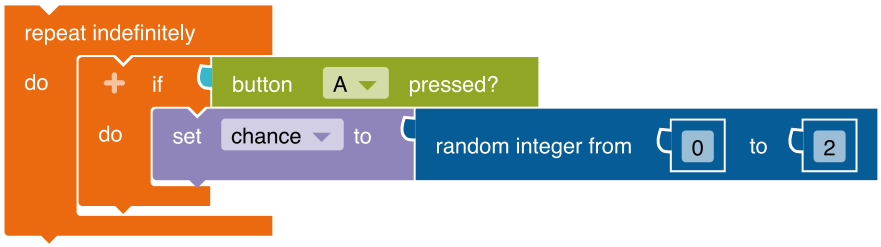


4 After pressing the Button A, an image and the corresponding text should be displayed:
To do this, an infinite **loop** is first inserted in the program.



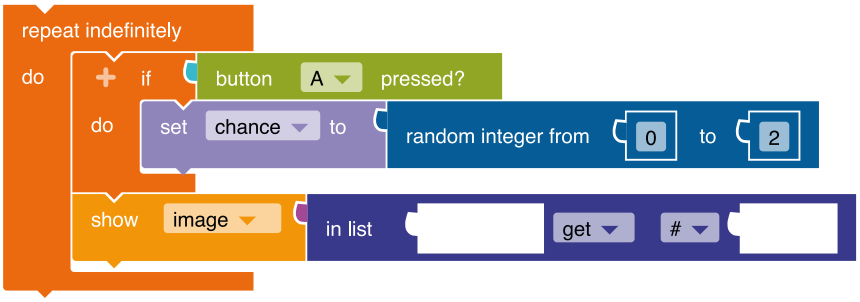
Add a **branch** to the infinite loop to check whether the user is pressing the button A.
Select the appropriate blocks from the categories **Control** and **Sensors**.

5 The image to be displayed and the associated text should be selected at random:
Select from the category **Variables** the block "Write *chance*" and insert it into the branch.
Append the block "random integer value" from the category **Mathematics** to it. Change the limits for the random value to 0 for the lower and to 2 for the upper limit.
With these **commands** the *Calliope mini* generates a random number (0, 1, 2).
This generated number is written into the variable *chance*.
This means that the variable *chance* stores the value that was written into it.

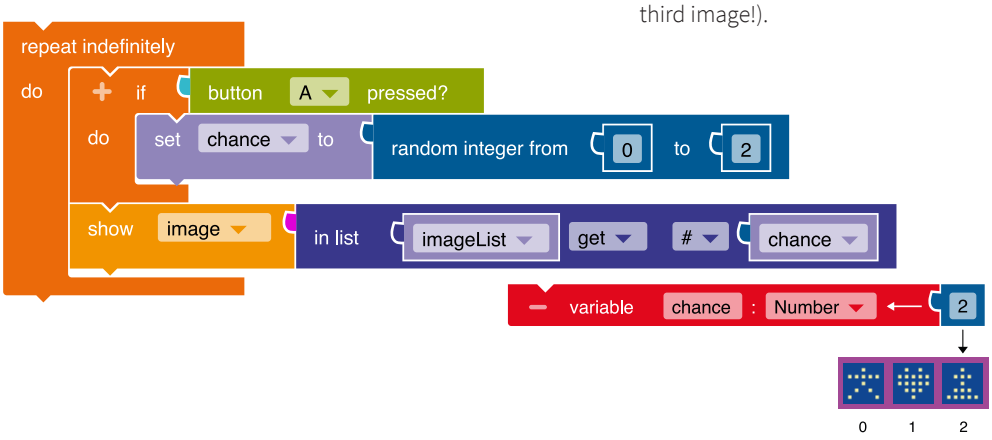


6 An image is to be displayed that is at the position in the list that is identified by the value of the variable *chance*:
The block "Show image" is used to display an image from the category **Action**.
Insert it under the branch.

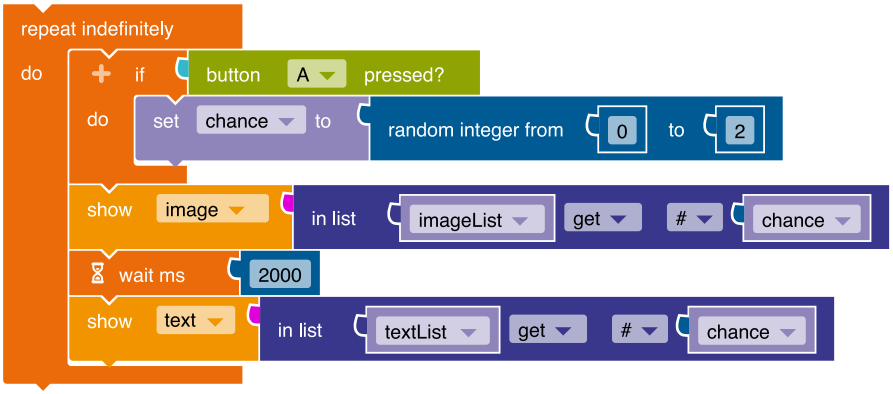
Then select from the category **List** the block "in list get #".
It is used to select an element of a list via its number (**Index**).
The first image in the list has the number (index) 0, the second 1 etc.



7 When selecting the list elements, it must be determined from which list they come:
Insert the variable named *imageList* from the category **Variables** into the gap after "in list". Also from the category **Variables**, take the variable *chance* and insert it into the space after "#". This selects the image that is at the position in the list whose number corresponds to the value of the variable *chance*. For example, if the variable has the value "2", then the image is selected which is at position "2" in the list (the third image!).



8 A word that matches the image displayed should also be output:
Select from the category **Action** the block "Show text" and insert it under the block "Show image". Make a copy of the list selection. Change the copied line so that *Text list* is in the copied block. To do this, click on the word *Image list* and select the word *Text list* in the drop-down menu.
Add the block "from the list" to the "Show text" block. Note that this will only work if you have selected the word *textList*. The new line shows a text from the text list which is selected by the value of the variable *chance*.
In order to be able to retrieve the same value here again, it must first be temporarily stored in the variable. If the selection value used for the text selection were different, the text and image would no longer match.
Add a "Wait ms" block between the show blocks so that the image is visible for two seconds.



9 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

The students first select images from *NEPO* ® and give them the appropriate words. Then they start working in pairs, swap their circuit boards and begin to write a story based on their partner's terms. Due to the random selection, the order in which the terms appear cannot be foreseen.

In the further course of teaching you can design your own images and add terms to them.

Expansion possibilities

The program can be expanded in such a way that, in addition to the images and texts, random colours are also displayed, which must be included in the story to be written.

The *Calliope mini* as a spelling trainer

The exercise

In this exercise the *Calliope mini* is programmed as a spelling trainer. If it is turned upside down and then returned to its original position, a word appears on the LED screen as a ticker. One letter of this word is replaced by an underscore. The word is followed by two letters, one of which is the correct supplementary letter. If the first letter is correct, the Button A must be pressed; if the second is correct, the Button B. A tick (correct) or a cross (incorrect) indicates whether the entry is correct or incorrect. Entering the correct solution is reinforced by a higher tone and the green RGB LED lighting up, an incorrect entry is accompanied by a lower tone and the red RGB LED lighting up (incorrect). Turning the *Calliope mini* upside down and back to the upright position makes the next word appear.
The words displayed can be changed and added to as required.

Subject matter

English

The students practice words with the *Calliope mini* and check them for their spelling accuracy. During the programming process, you can put together individual words that are not yet fully saved and adapt them to the current state of knowledge.

Position in the syllabus

Content-related skills

The students can write correctly in the field *Write texts*. They make use of the regularities of standardised spelling, check their texts for spelling accuracy and apply spelling patterns and strategies.

Process-related skills

The students can use spelling strategies in the field of *Writing*, they are error-sensitive and have a feel for spelling and use spelling programs in electronic media as a proofing aid.

Requirements

Programming focus:

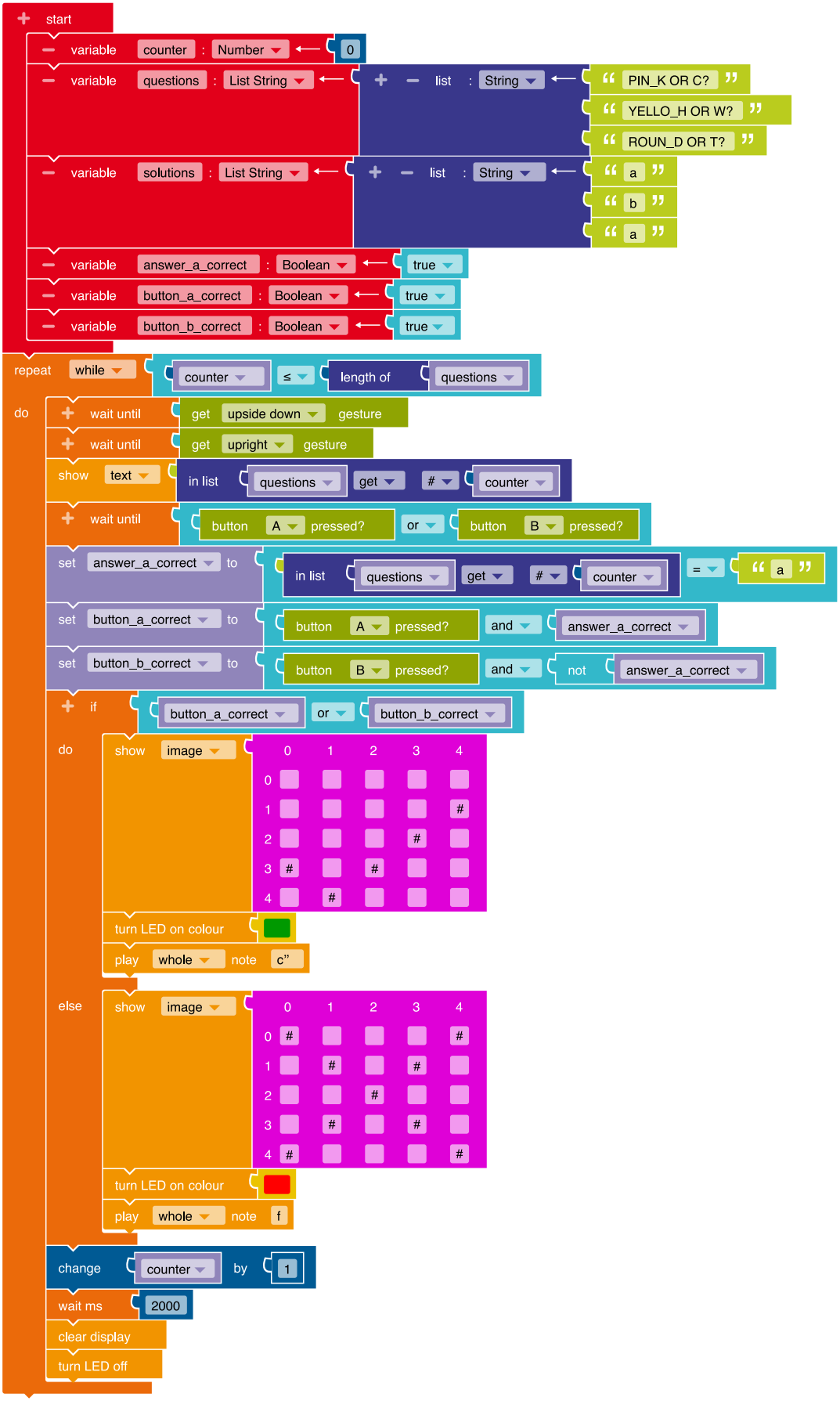
- Input via buttons, output via LED screen, RGB LED and loudspeaker
- Structures: conditional loop, conditional waiting, branch
- Formulate conditions
- work with lists
  (Index ▸ Position of a list element)
- *NEPO*® categories used: Action, Sensors, controls, logic, mathematics, lists, variables

Programming difficulty:

- Advanced level
- *NEPO*® Level: Expert

The code

This is what the code of the finished program looks like. It is defined step by step below.



Steps to create the program for the spelling trainer

1

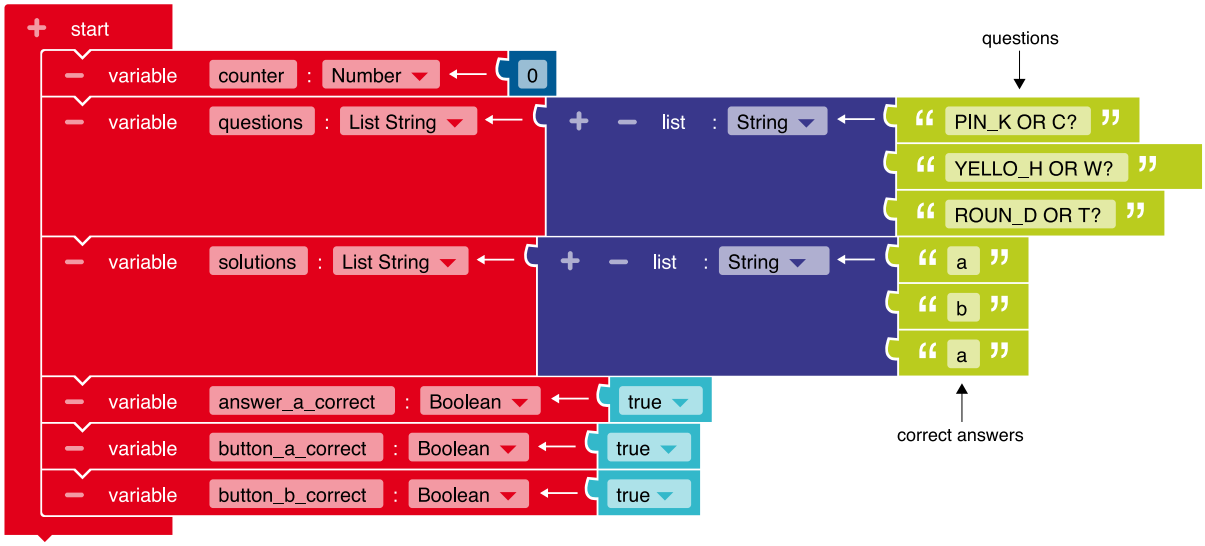
Practice words and their correct spelling should be determined for the spelling trainer:
To create a new **Variable**, click on the "+" to the left of "Start". A new block appears. Another **string** is also added by clicking on the "+" in the blue list block.

A total of six **variables** are created:

- one with the name *counter* from the **data type** "Number".
- two further **variables** with the names *question* and *answers* of the data type "**List** Strings".

Enter the question words (freely selectable) and the correct answer ("a" or "b") in the light green text fields.
Note: The underscore in the question words stands for the letter to be used.

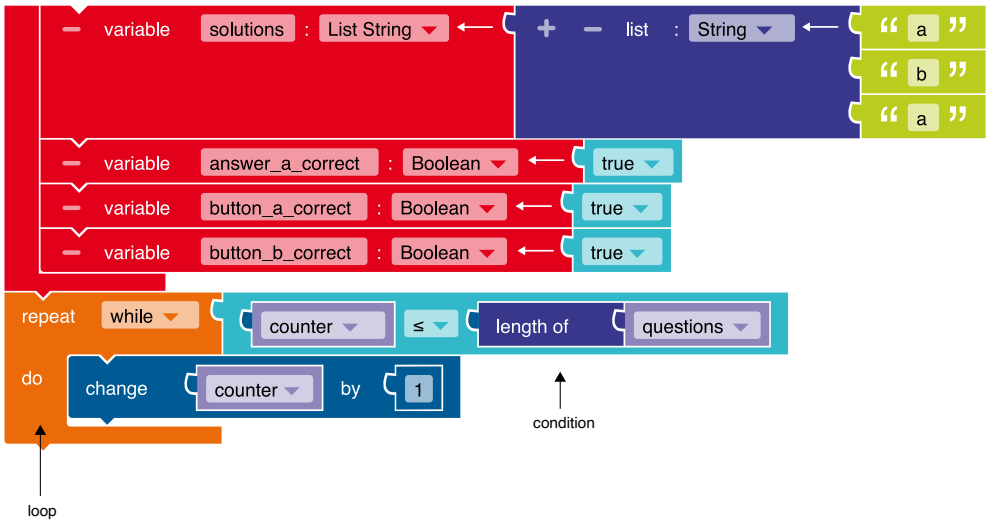
Finally, you need three **variables** of the data type "Boolean" with the names *answer_a_correct* , *button_a_correct* and *button_b_correct*.
All three receive the **truth value** "true" from the category **Logic**.



2

A **loop** (repeated **command**) is created, where the processing of each question represents a loop pass:
Select the block "repeat while/do" from the category **Control › Loop** and add it to the red variable blocks.
This loop should be repeated as long as there are questions in the list.
The **variable** *counter* initially has the value "0" and in the last loop pass it has the value "2" for three questions. This is because list indices always start with the **index** "0".

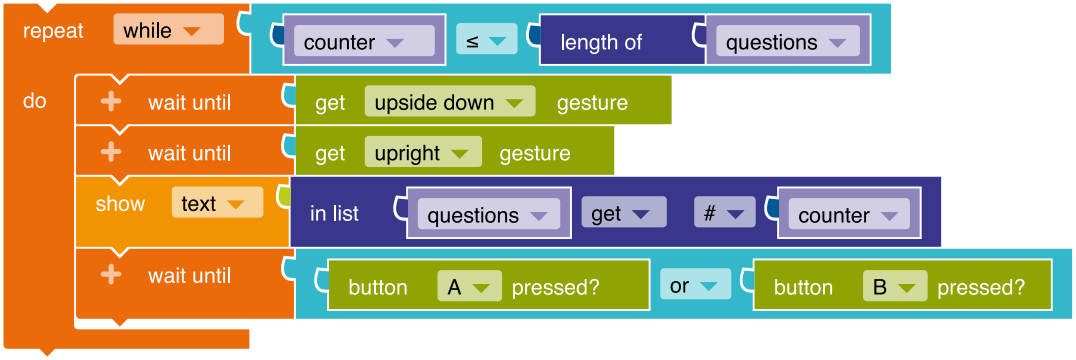
For the loops **condition** you need from the category **Logic** the **comparison** block "≤", from the category **Variables** the blocks "counter" and "questions" and from the category **Lists** the block "Length of". From the category **Mathematics**, insert the block "change by" into the loop, in which you enter the numerical value "1".



3 The player takes control of the spelling trainer using the following instructions:
Select the "Wait until" block from the category **Control** and insert it three times in the loop in front of the "Increase counter by 1" block.
At the first two "Wait until" blocks you dock the blocks "Get upside down gesture" and "Get upright gesture" from the category **Sensors**.

At the third "Wait until" block, from the category **Logic** add the block  . Select in the **drop-down menu** "or". Fill in the two gaps with the blocks "Button A pressed?" and "Button B pressed?" from the category **Sensors**.

Now the *Calliope mini* is ready for the next question:
To show these randomly, select the block "Show text" from the category **Action** and add the block "from the list" questions "in list get #" from the category **Lists**. You fill the two gaps with the blocks "questions" and "counters" from the category **Variables**, since these are used as a list index.



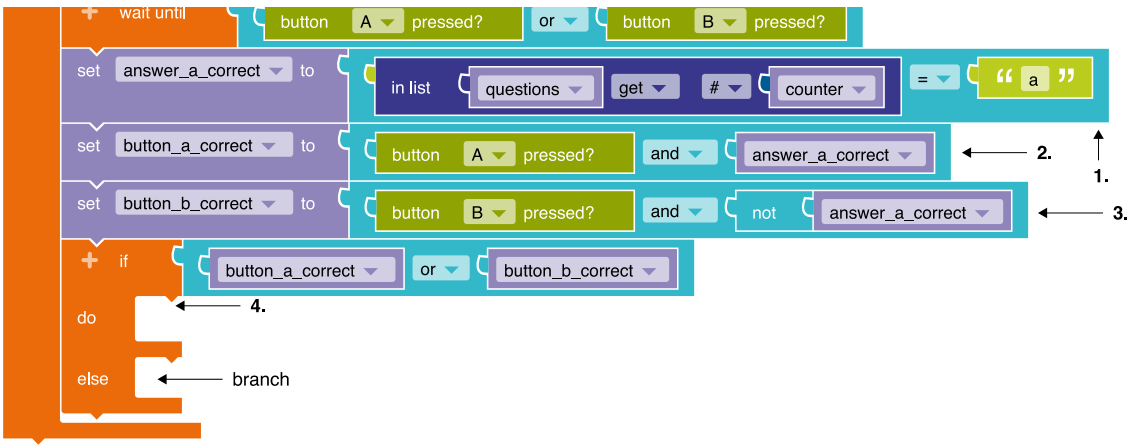
4 The next four steps check whether the button you pressed matches the correct solution:
As preparation, select three "write" blocks from the category **Variables**, within which you select the variables *answer_a_correct* , *button_a_correct* and *button_a_correct*. To the first "Write" block, add from the category **Logic** the block  , to the other two, the block  .

1. Is the entry in the "solutions" list equal to "a"?
   Add in the first condition the block "in list get #" from the category **Lists**. Fill the left gap with the "solutions" block, the right with the "counter" block from the category **Variables** . On the right side of the equal sign insert the letter "a" from the category **Text**.
2. If the answer "a" is correct, was the button "A" also pressed?
3. If the answer "a" is incorrect, answer "b" must be correct. Is then the "B" button pressed?

Step **2** and **3**:
In the two following **conditions** insert from the category **Sensors** "Button A pressed?" and "Button B pressed?" on the left side and the **Variable** *answer_a_correct* is inserted twice on the right side. For the lower condition **3** the **variable** *answer_a_correct* is enclosed by the negation block "not" from the category **Logic**.

4. If the result of one of the previous two questions (step 2 or 3) is true, the player has given a correct answer.

This is followed by a **branch** from the category **Control › Decisions** with the block "if/do/else". Dock the condition block  from the category **Logic** to this. The "and" is changed in the **drop-down menu** to "or" and add to the left and right of the "or" the blocks *button_a_correct* as well as *button_b_correct* from the category **Variables** .

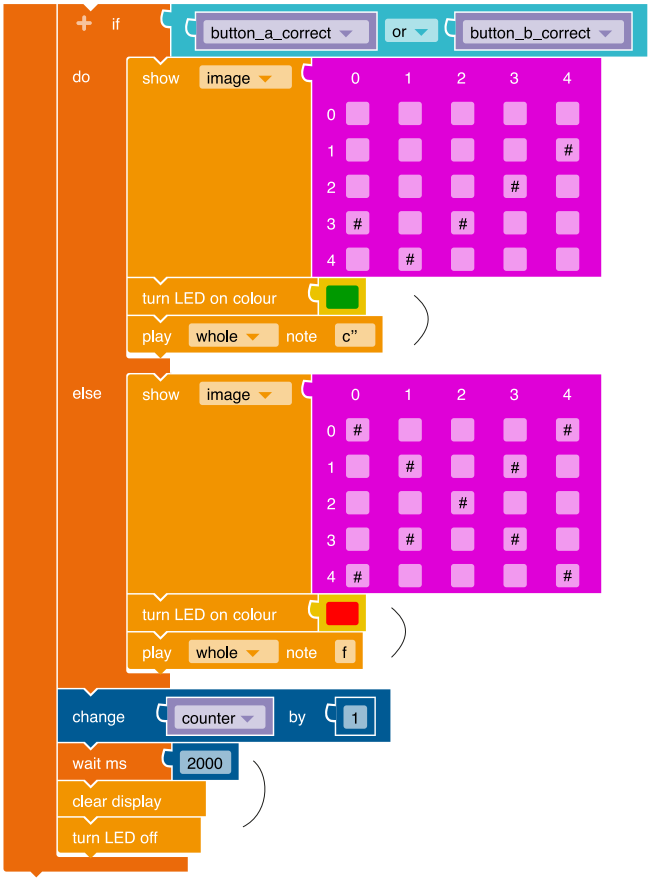


5 There is now output that matches the correct and the incorrect answer in the form of an image:
If the comparison of the button input matches the correct answer, you can now use the branch after "do" via the category **Action › Display** and the block "Show image" to insert a hook for example.
After "else", if a button that did not match the correct answer was pressed, an image is added for the **LED screen** with an "X" for example.

As additional feedback to the user, the **RGB-LED** should light up and a tone should be heard.

Between each question the LED screen must be cleared and the RGB LED turned off:
At these points you can add from the category **Action › Light status** the block "Turn LED on colour" and from **Action › Sound** the block "Play whole note C" for the additional output for the RGB-LED and the sound output.

Between the questions, i.e. after the block "if/do/else", the LED screen is deleted after a waiting period (**Control › Wait**) of two seconds (= 2000 milliseconds) and the RGB-LED turns off again (**Action › Display** and **Action › Light status**).



6 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

- Any spelling exercises in which individual letters make the word difficult to write, for example spelling phenomena such as the hardening of the final sound
- Finding such words in texts or exercises, for example from the textbook, and adding these words to the list of questions in the spelling trainer

Expansion possibilities

- an extension would be to count the correct answers as a further **variable** *points*
- likewise, the respective word could be displayed in the correct spelling after each question. Possibly also in the plural or in the infinitive, for example, to justify the correct spelling
- Similar to the program example "Morse code with the *Calliope mini*", questions and answers can be transmitted wirelessly between two *Calliope mini* (Category **Messages** ▸ Block "Send message")

The Calliope mini as a random generator

The exercise

In the following exercise, the *Calliope mini* is programmed to be a random generator (number dice). Pressing a button will output a random number between 1 and 6. In a further step, the dice can be programmed in such a way that the respective number image of the dice is displayed instead of the figure.

Subject matter

Mathematics

The students program the *Calliope mini* as a random generator (hexagonal dice). They can use it to carry out simple random experiments and assess, describe and compare the probability of events.

Position in the syllabus

Content-related skills

In the field of *data, frequency and probability*, the students can carry out simple random experiments as well as assess, describe (certain, possible, impossible) and compare the probability of events in simple random experiments by using the programmed *Calliope mini* dice in a playful and practical way for random experiments.

Process-related skills

In the field of *Reasoning*, the students can ask questions and express assumptions as well as look for reasons (also of regularities).

Requirements

Programming focus:

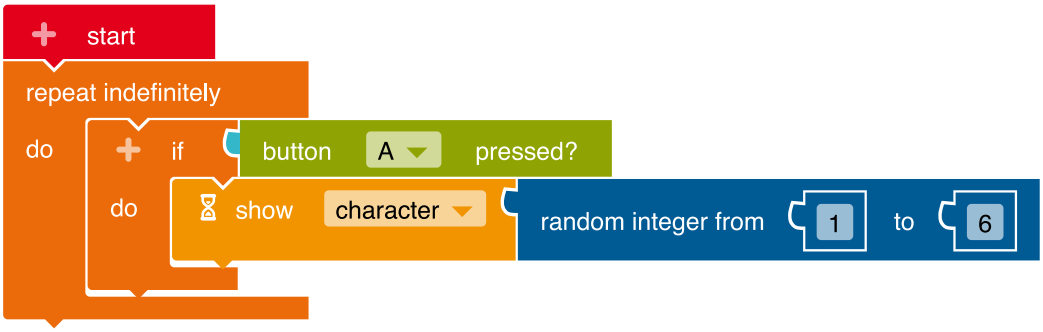
- Input via buttons, output via LED screen
- Structures: conditional command, loop
- Formulate conditions
- work with variables (code 2)
- Compare input values; Logic (code 2)
- *NEPO*® categories used: Action, sensors, controls, logic, mathematics, variables

Programming difficulty:

- Beginner level
- *NEPO*® Level: Expert

The Code 1: Programming a number dice

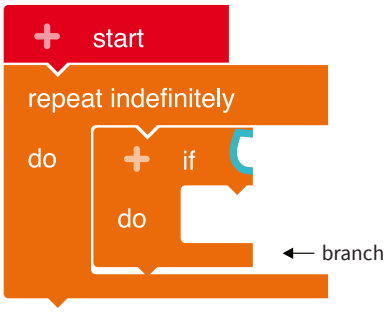
This is what the code of the finished program looks like. This is defined step by step below.



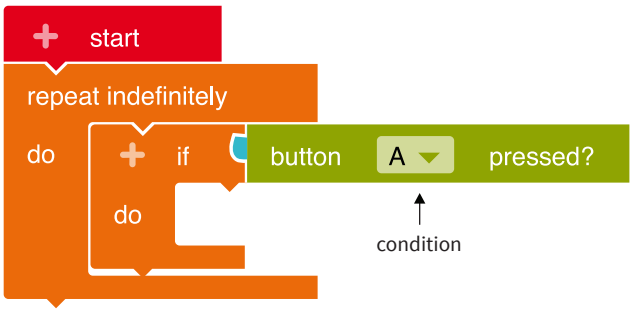
Steps to create the program for a number dice

1. So that an infinite number of random numbers can be output, an infinite **loop** is required:
Select the "Repeat indefinitely/do" block from the category **Control › Loop** and add it to the "Start" block.

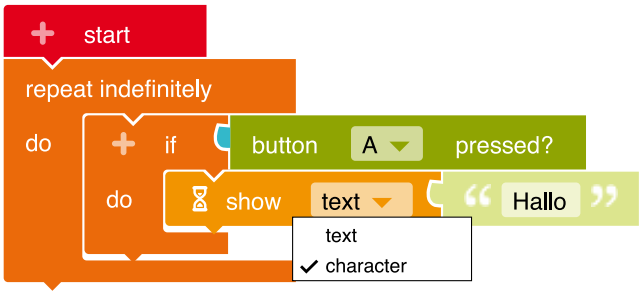
2. When Button A is pressed (*i*), a random number should be output (*do*):
For this you need a **branch**. Select the block "if/do" from the category **Control › Decisions** and paste it into the loop.



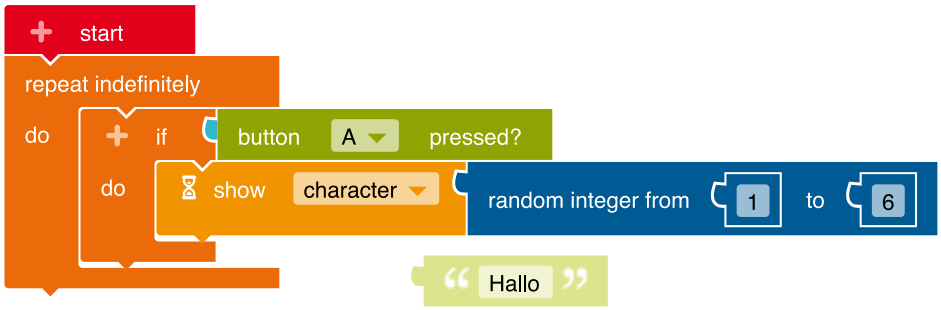
3. The desired **input** option (here via the **sensor** "Button A") is selected:
Select the block "Button A pressed?" from the category **Sensors** and add it to the branch as a **condition** (blue area).



4. The random number should be displayed on the **LED screen**:
Select the block **Action › Display** the block "Show text" and click in the **drop-down menu** on "Character".



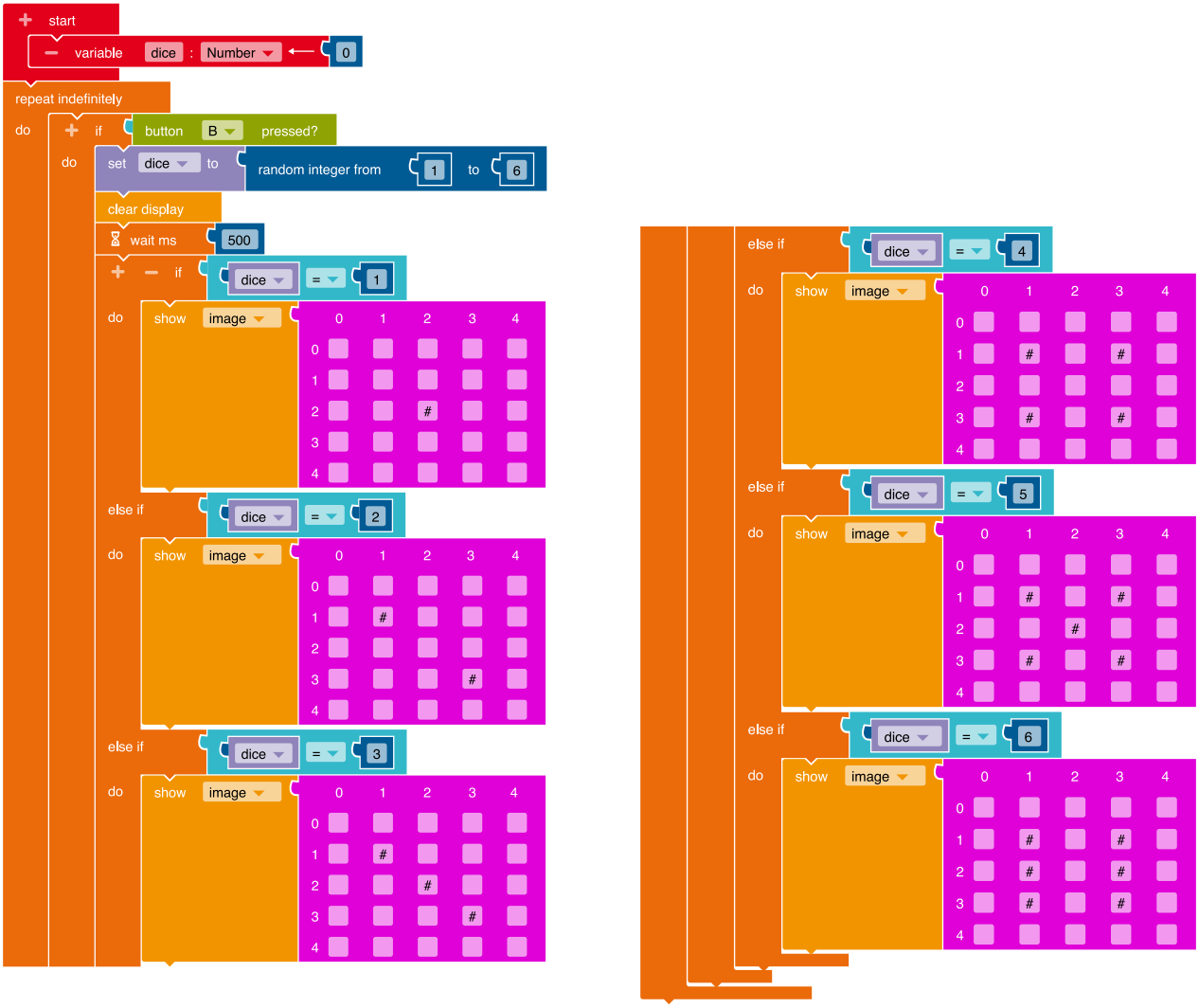
5. In the last step, the value of the **random** number must be specified:
Replace the text "Hello" attached to the block "Show characters" with the block "random integer from 1 to 100" from the category **Mathematics**. Replace the value 100 with the number 6 (hexagonal dice) by clicking on the 100 and entering the number 6 on the keyboard.



6. Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

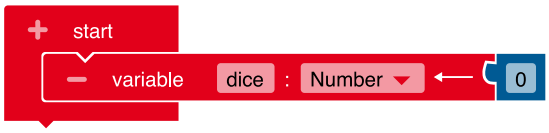
The code 2: Programming a dice with dots

This is what the code of the finished program looks like. This is defined step by step below.

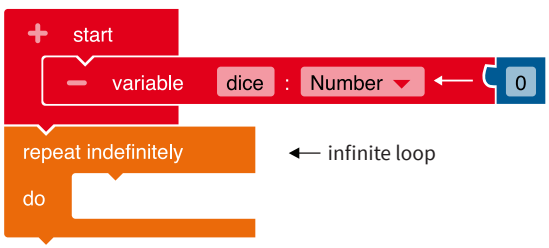


Steps to create the program for a dice with dots

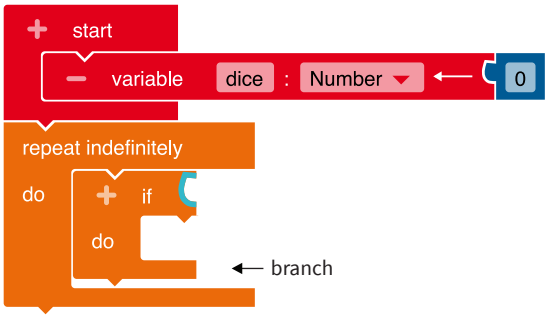
1 So that new numbers can be output again and again, a **variable** must be created and defined:
To create a new variable, click on the "+" to the left of "Start". A new block appears. Define the name (*dice*) and **data type** (here: number) of the variable. To do this, click on the relevant field.



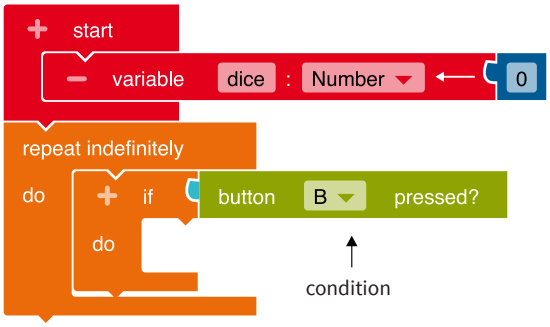
2 Unlimited random numbers should be able to be output:
For this you need an infinite **loop**.
Select the block "repeat indefinitely/do" from the category **Control ‣ Decisions** and add it to the "Start" block.



3 When the Button B is pressed, a random number is to be output:
To establish this **condition**, you need a **branch**.
Select the block "if / do" **Control › Decisions** and insert it into the loop.

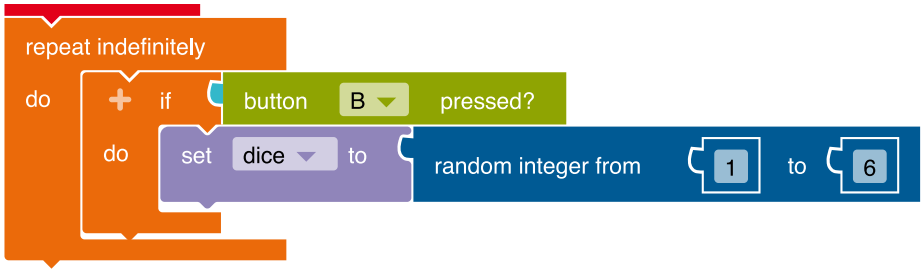


4 The desired input option (here via the sensor "Button B") is selected: Select the block "Button B pressed?" from the category **Sensors** and add it to the branch as a **condition** (blue area).



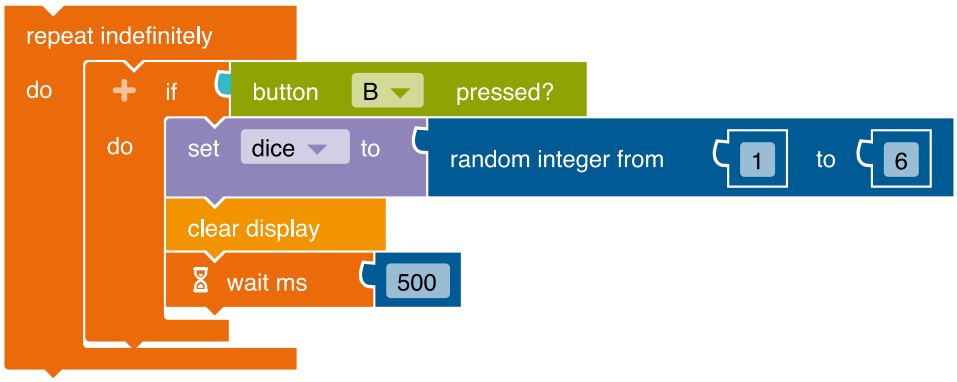
5 The value of the variable ***random*** *number* must be defined:

Select from the category **Variables** the block "set *dice*". Place the block "random integer from 1 to 100" from the category **Mathematics** and replace the value 100 with the number 6. To do this, click on 100 and enter the number 6 on the keyboard.



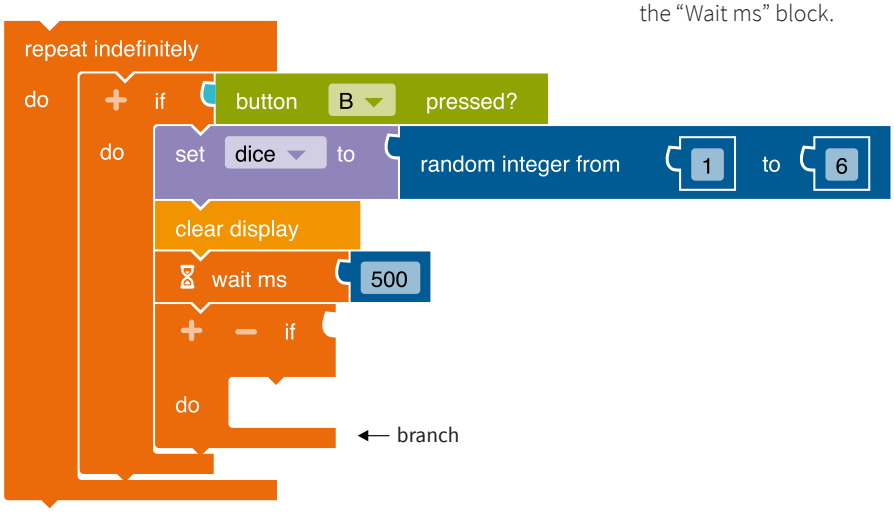
6 A better readability of the random numbers should be guaranteed:
Delete the screen content by inserting the block "Clear display" from the category **Action › Display**.

Then insert a pause. Select the block "Wait ms" from the category **Control › Wait** and define the length of the pause (1000 ms = 1 second).



7 The random values must be linked to the dot images of the dice:

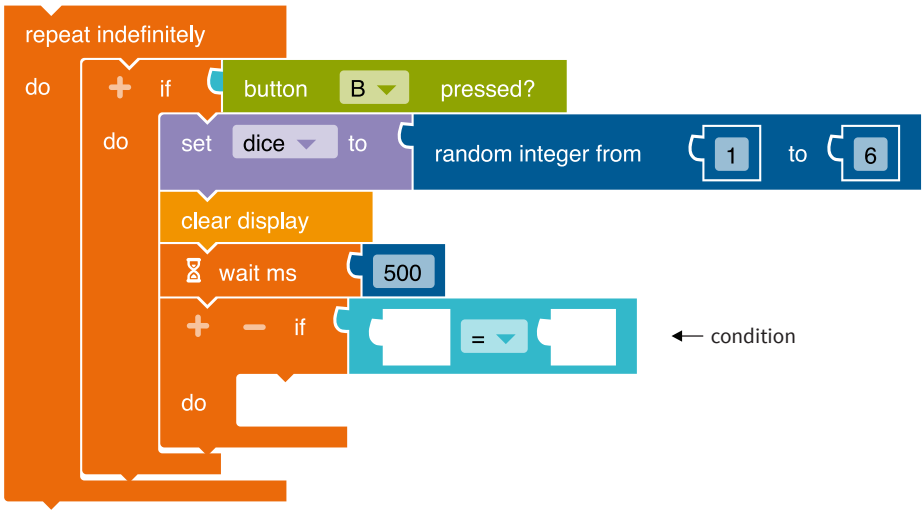
Another branch is needed for this. Select the block “if/do” from the category **Control › Decisions** and append it to the “Wait ms” block.



8 The random value must be queried so that the corresponding dot image can be displayed on the LED screen:

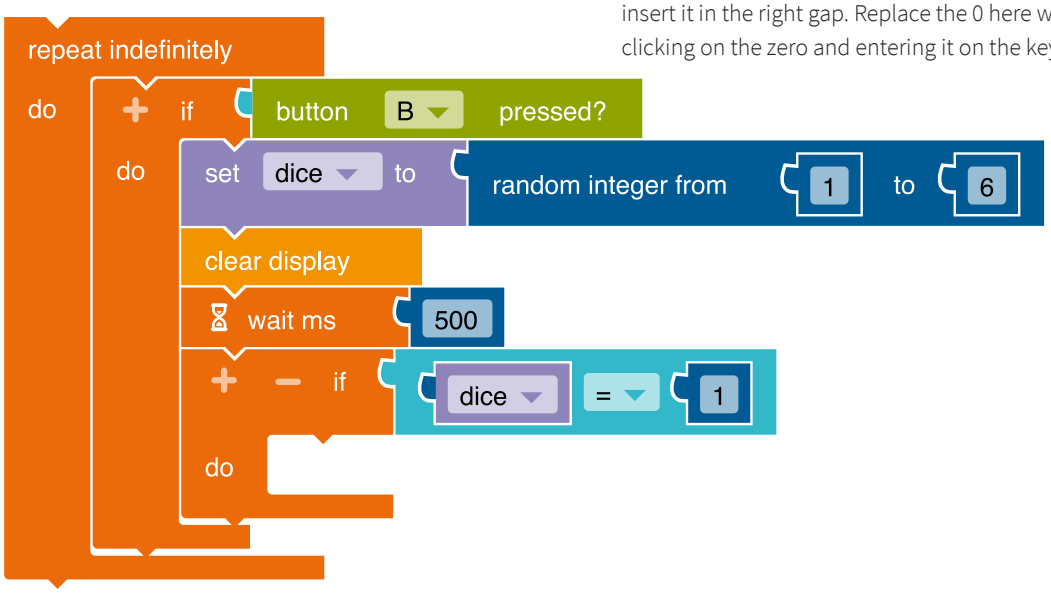
For this you need a **comparison** between two numbers.

Select from the category **Logic** the block  and add it to the branch as a condition.



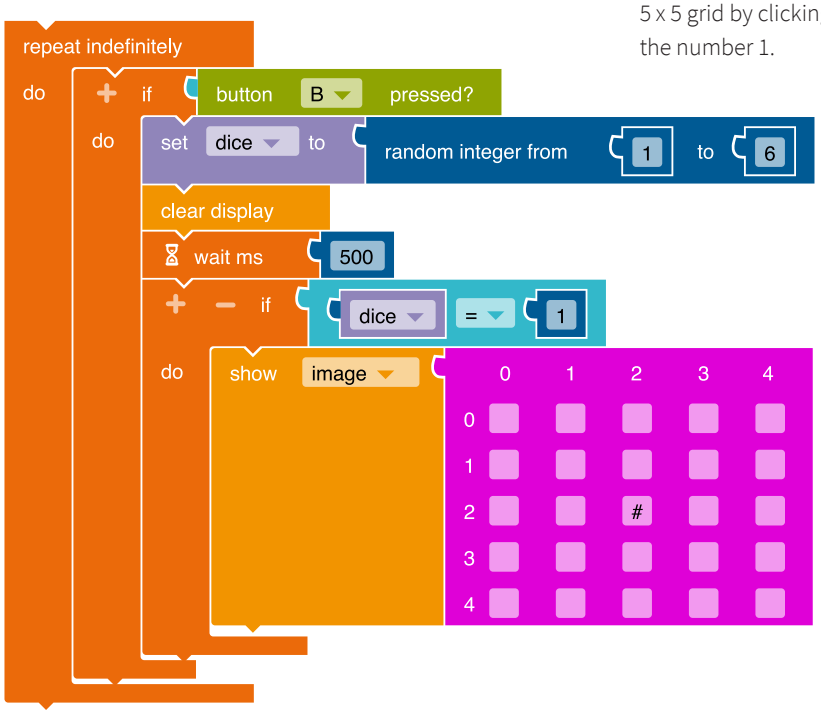
9 The random number from the variable *dice* is now compared with the number 1:

Select from the category **Variables** the block “*Dice*” and drag the variable into the left gap of the logic block. Take the block “0” from the category **Mathematics** and insert it in the right gap. Replace the 0 here with a 1 by clicking on the zero and entering it on the keyboard.



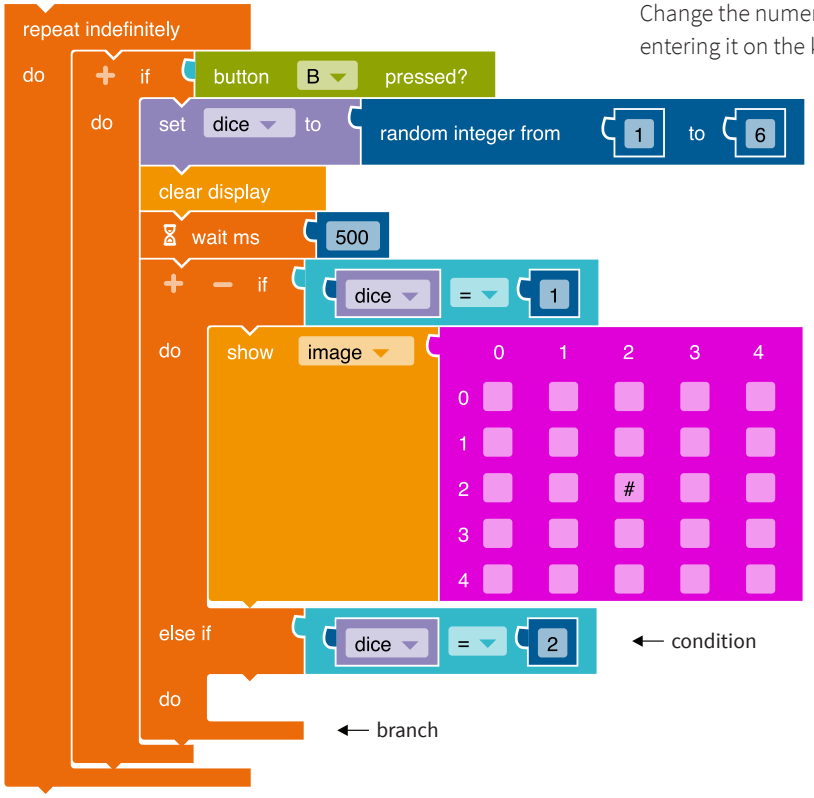
10 The dot image for number 1 is displayed on the LED screen:

Select the block “Show image” from the category **Action › Display** and mark the point in the middle of the 5 x 5 grid by clicking on the field that is required to display the number 1.

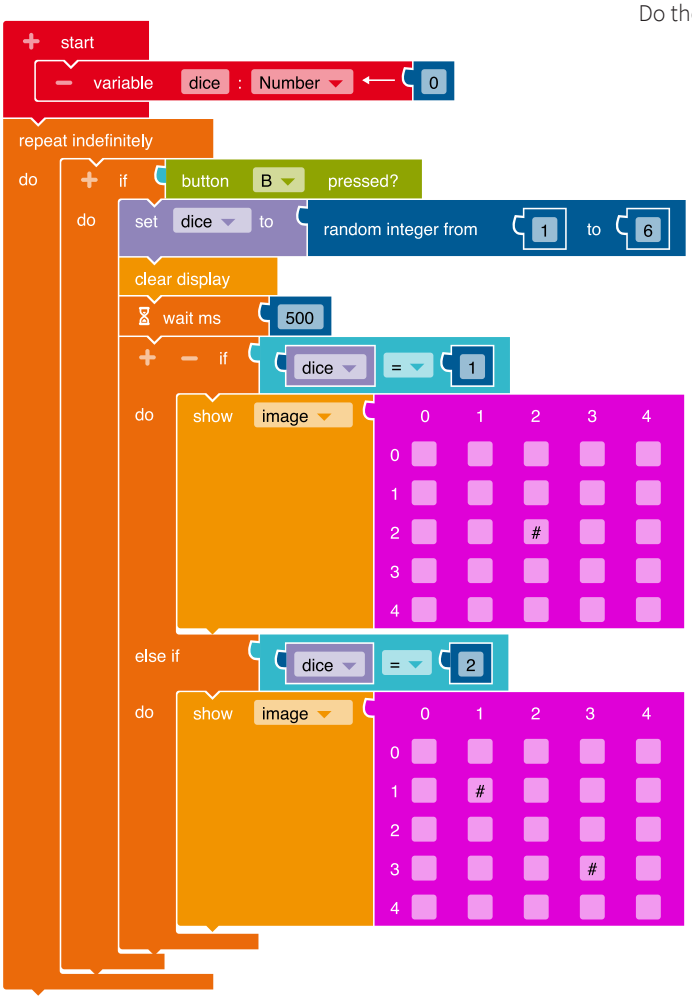


11 The dot image for number 2 should be linked and displayed on the LED screen:
Now click on the “+” next to the “if” in the branch.

Another branch appears.
Now copy the complete **Logic** block from step 10 by clicking with the right mouse button on the block to be copied and selecting “copy” and append it to the branch. Change the numeric value to 2 by clicking on the one and entering it on the keyboard.



12 The dot image for number 2 is displayed:
Copy the block **Action › Display** "Show image" and mark the corresponding boxes for the display of the number 2 by clicking the corresponding fields.
Do the same for the remaining four sides of the dice.



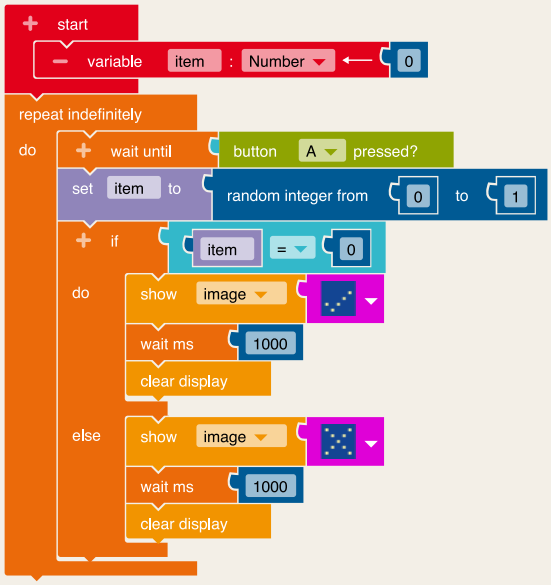
Notes and information

Implementation scenarios in class

- Compare your own experience of throwing the dice (luck, bad luck) and findings from test series with a large number of dice throws.
- Make assumptions about random experiments and their outcome.
- Create frequency lists working in pairs (first with one, then with two *Calliope mini* dices); describe the results and try to justify them
- Compare the digital dice with real dice and use it
- Store further platonic solids (hexagonal or multi-faceted solids) in the program code and examine them

Expansion possibilities

The *Calliope mini* can be programmed into a mini oracle in a similar way. When you click on the Button A, either a tick or a cross, "yes" or "no", a sad or a happy face, etc. appears at random.



The Calliope mini as a 1 x 1 mental arithmetic trainer

The exercise

In this exercise, the *Calliope mini* is programmed to become a 1 x 1 mental arithmetic trainer. For example, if you click the Button A, a multiplication problem is displayed. For example, if Button B is pressed, the result of this task will appear on the LED screen. Programming can be done in two steps:

Code 1: Programming of individual multiplication problems:
In a first step, one or more self-defined multiplication tasks are programmed, whereby different sensors (e.g. Button A or B, Touch-Pin pressed, shaking, position upside down etc.) can be used.

Limits of the code
With code 1, the program does not perform any arithmetic operations. Instead, the correct result must have been calculated and stored beforehand. This means that work with the 1 x 1 mental arithmetic trainer will very soon be used up and the user quickly notices that a program expansion is necessary.

Code 2: Programming of the randomly generated 1 x 1 calculator trainer:
In the next step, a programming code must be created in which the *Calliope-mini* actively calculates itself. In doing so, new multiplication tasks should always be given out at random and the corresponding correct solutions displayed.

Subject matter

Mathematics
The students program a 1 x 1 mental arithmetic trainer that outputs the two factors of a multiplication problem at random and shows the correct solution at the push of a button.

Position in the syllabus

Content-related skills
In the field of *Numbers and operations* , the students can call up the basic arithmetic tasks from memory by first programming a *Calliope mini* arithmetic trainer and then practicing the arithmetic tasks as a user.

Process-related skills
In the field of *Representation*, the students can transfer one representation into another (in the programming language of the editor).

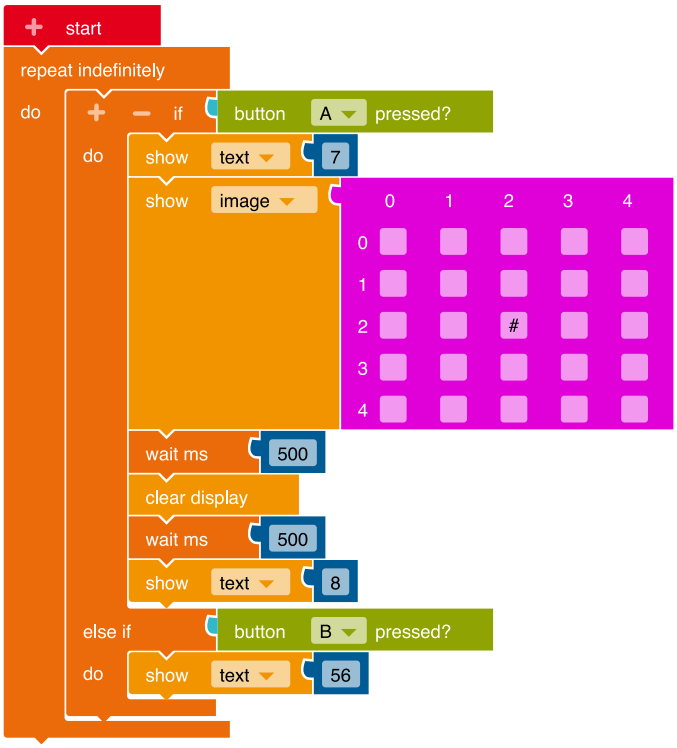
Requirements

Programming focus:
- use different input options
- Output of text (number) and image
- Structures: Loop, conditional wait
- Formulate conditions
- work with variables (Code 2)
- *NEPO*® categories used: Control, action, mathematics, sensors, variables (only code 2)

Programming difficulty:
- Medium level
- *NEPO*® Level: Beginner (Code 1) – Expert (Code 2)

The Code 1: Programming of individual multiplication problems

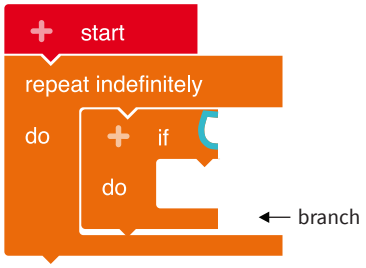
This is what the code of the finished program looks like. It is defined step by step below.



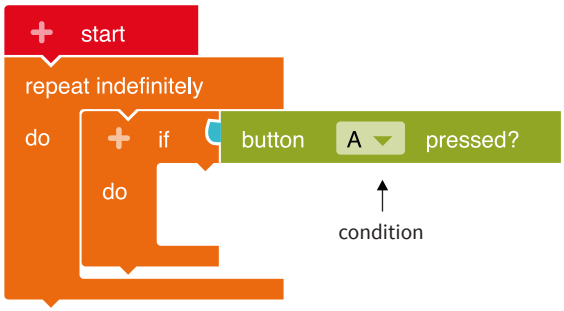
Steps for creating the program for individual multiplication tasks

1 So that the multiplication problem can be displayed infinitely often on the *Calliope mini*, an infinite **loop** is created at the beginning:
Select the block "repeat indefinitely/do" from the category **Control** and add it to the "Start" block.

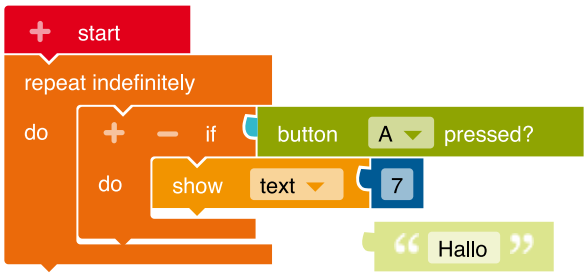
2 When the Button A is pressed (*if*), the multiplication problem should be displayed (*do*):
To establish this **condition**, you need a **branch**.
Select from the category **Control** the block "if/do" and insert this into the loop.



3 The desired input option (here via the sensor "Button A") is selected:
Select the block "Button A pressed?" from the category **Sensors** and add it as a condition (blue area) to the branch.

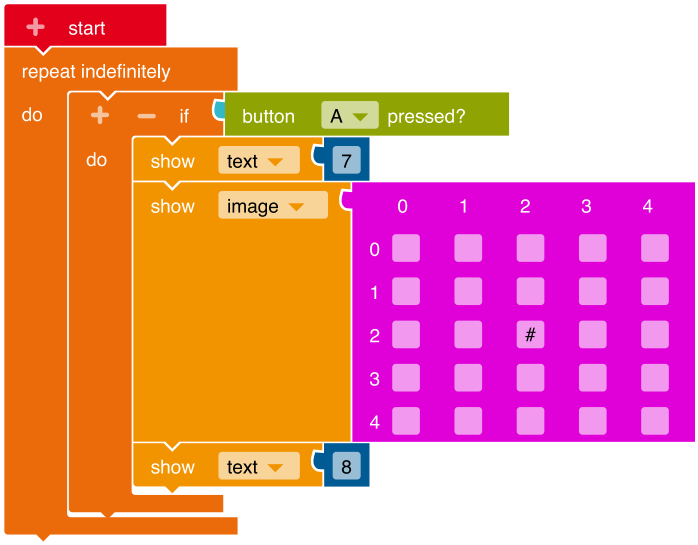


4 The 1st factor of the multiplication problem should be displayed on the **LED screen** :
Select the block "Show text" from the category **Action** and insert it into the branch. Replace the text "Hello" with the block "0" from the category **Mathematics**. Add any number between 0 and 10 (▸ factor 1) by clicking on the zero and entering it on the keyboard.



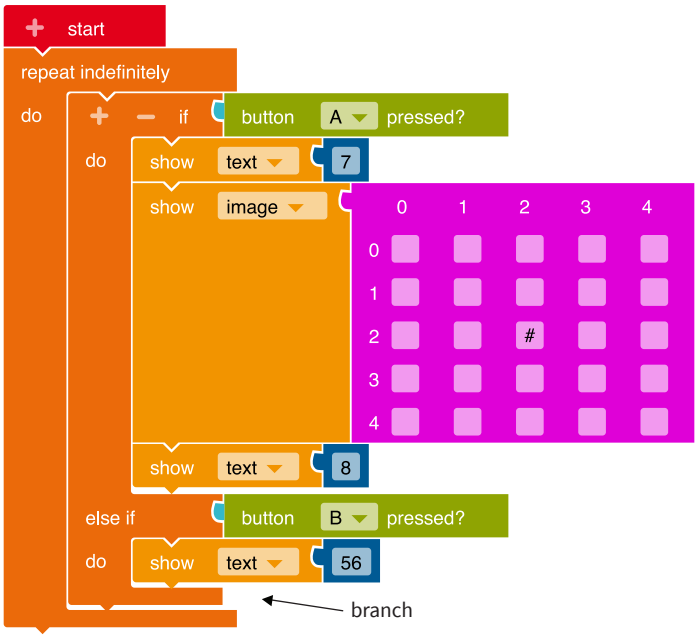
5 The multiplication sign should be displayed:
Select the block "Show image" from the category **Action** and mark the box in the middle. This point appears on the LED screen as a multiplication sign.

Then enter a second number (▸ factor 2). Proceed as in step 4.



6 When button B is pressed (*if*), the result of the multiplication problem should be displayed on the LED screen (*do*): Another branch is needed for this.
In the first branch click on the "+" next to the "if".

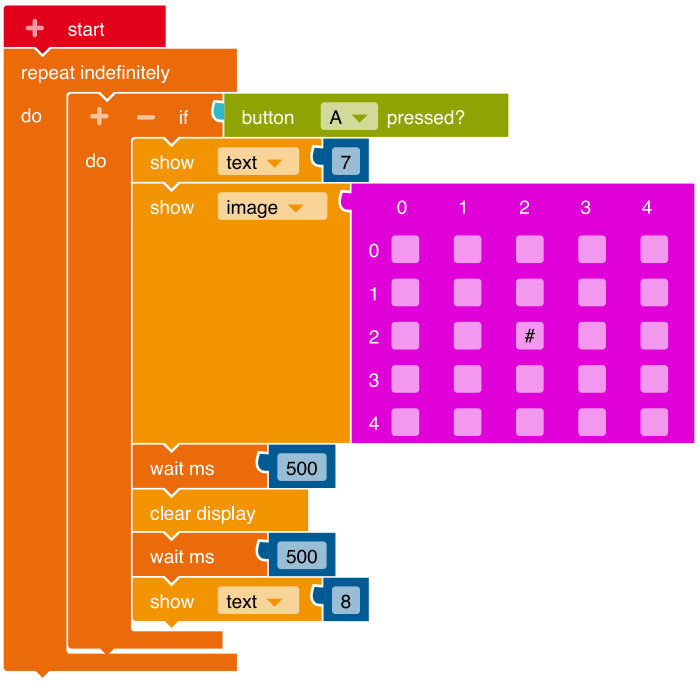
First select the block "Button B pressed?" from the category **Sensors** and add it as a condition to the branch. Then add the "Show text" block from the category **Action**. Replace the text "Hello" with the block "0" from the category **Mathematics**. Enter the correct result of the multiplication problem you have saved here.



7 When running the task, the multiplication sign is difficult to see. Adding **pauses** and clearing the LED screen increases readability:
Select the block "Wait ms" from the category **Control** and specify the length of the pause here.

To clear the screen content, select the "Clear display" block from the category **Action**.

Insert another pause and test which pause length guarantees good readability. (1000 ms = 1 second)



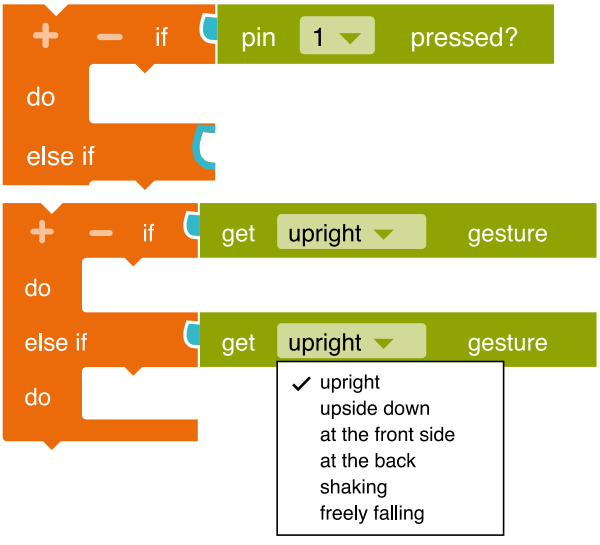
8 Expand the program so that several tasks can be practiced: Create further branches and select new conditions (**Sensors**), e.g.:

Sensors ▸ "Pin 1 pressed?"

Sensors ▸ "upright position active?"

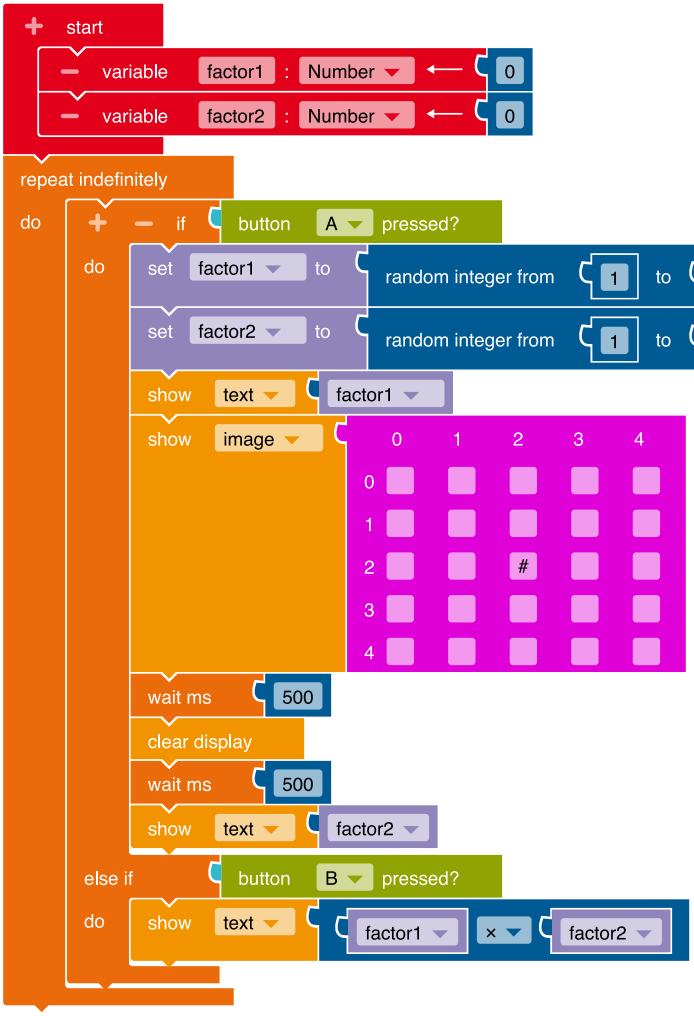
Proceed as described in steps 2–8.

9 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).



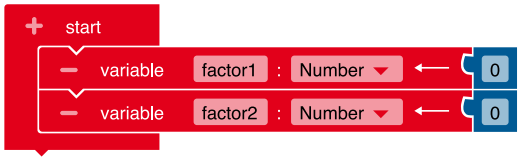
Code 2: Programming of the randomly generated 1 x 1 mental arithmetic trainer

This is what the code of the finished program looks like. It is defined step by step below.

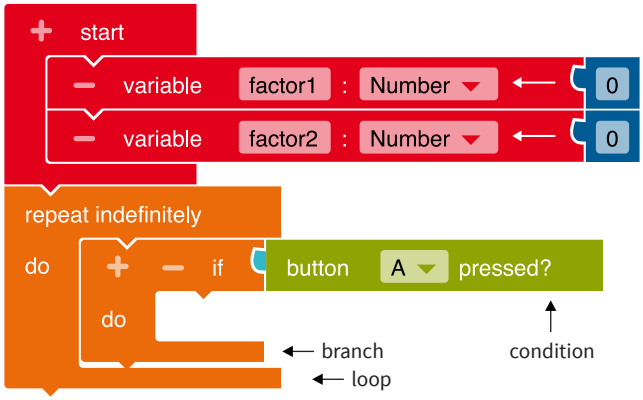


Steps to create the program for a randomly generated 1 x 1 mental arithmetic trainer

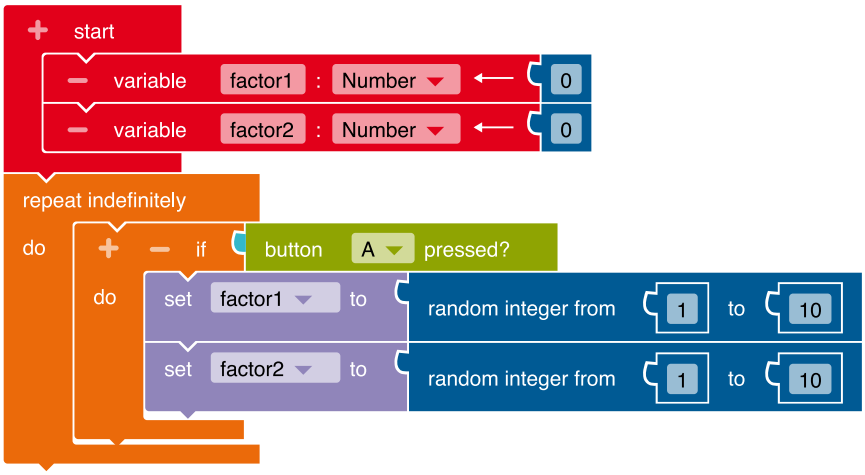
1 So that new tasks can be output again and again, two **variables** must first be created and defined:
You need variables for the two factors (*factor1* and *factor2*).
To create a new variable, click on the "+" to the left of "Start". A new block appears. Define the name (*factor1* / *factor2*) and data type (here: number) of the variable.



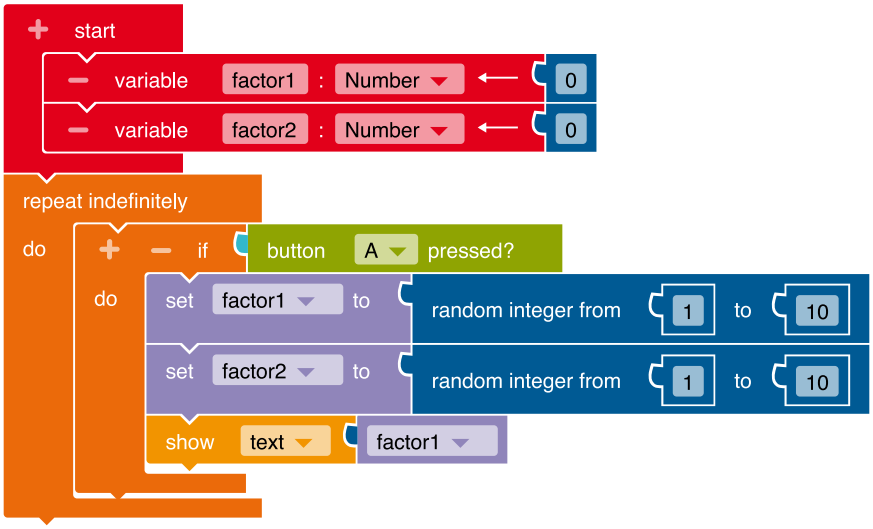
2 When the Button A is pressed (*if*), a multiplication problem generated by **random** should be displayed (*do*):
As described in Code 1, a **loop** and a **branch** are also required here:
Proceed as described in Code 1 (steps 1–3).



3 The value of the variable must be specified:
Select the block "Write *factor1*" from the category **Variables** and insert it into the branch. Place the block "random integer from 1 to 100" from the category **Mathematics** and replace the value 100 with the number 10. Do the same with *factor2*.

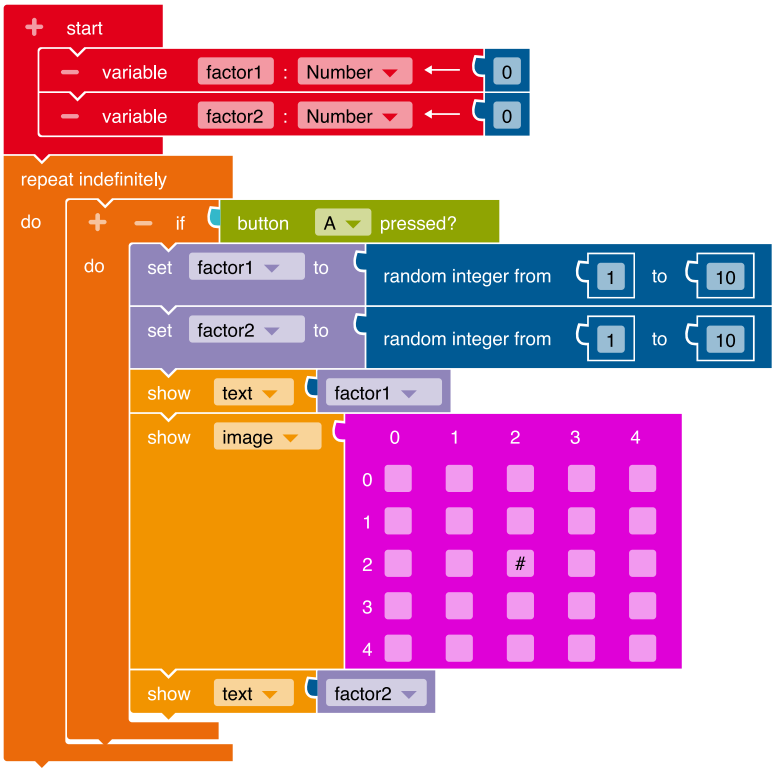


4 The 1st factor (*factor1*) should be displayed on the **LED screen**:
Select the block "Show text" from the category **Action ‣ Display** and replace the text "Hello" with the first **variable**. Select from the category **Variables** the block *factor1*.



5 The multiplication sign should be displayed on the LED screen:
Select the block "Show image" from the category **Action ‣ Display** and mark the box in the middle.

The 2nd factor (*factor2*) should be displayed on the LED screen:
Proceed as in step 4. Select from the category **Variables** the block "*factor2*".

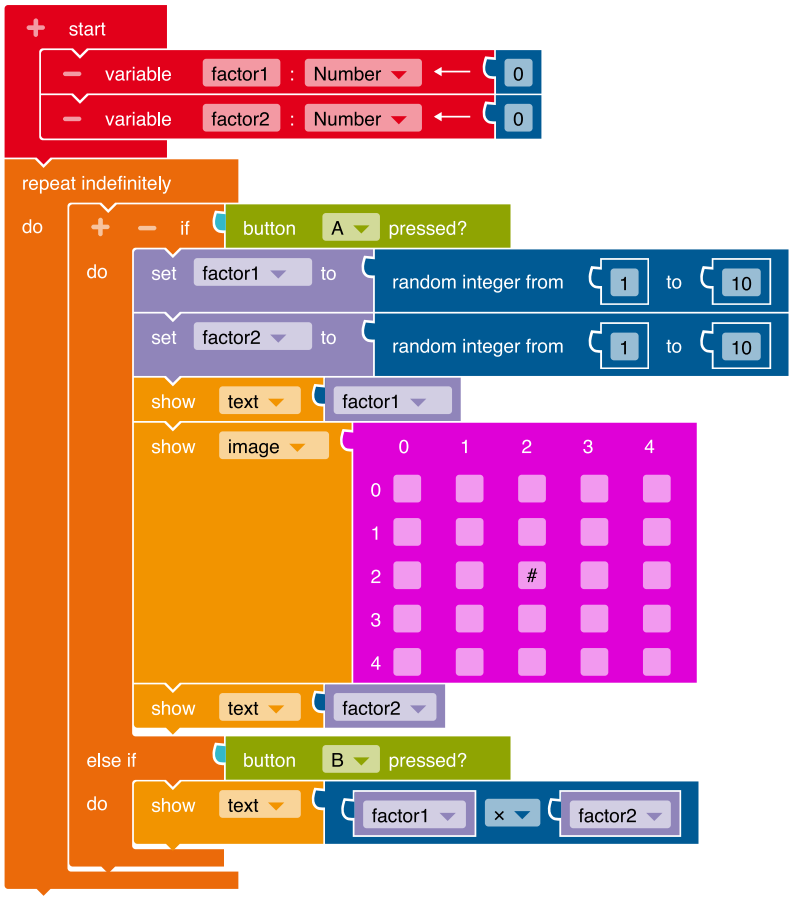


6 When Button B is pressed (*if*), the correct result of the randomly generated task should be displayed (*do*):
In the branch, click on the "+" next to the "if". Another branch appears. Select the block "Button B pressed?" from the category **Sensors** and add it as a condition to the branch.

The program needs the **command** that the product of the randomly generated factors is displayed on the LED screen:

Select "Show text" from the category **Action ‣ Display** and from the category **Mathematics** add the block .

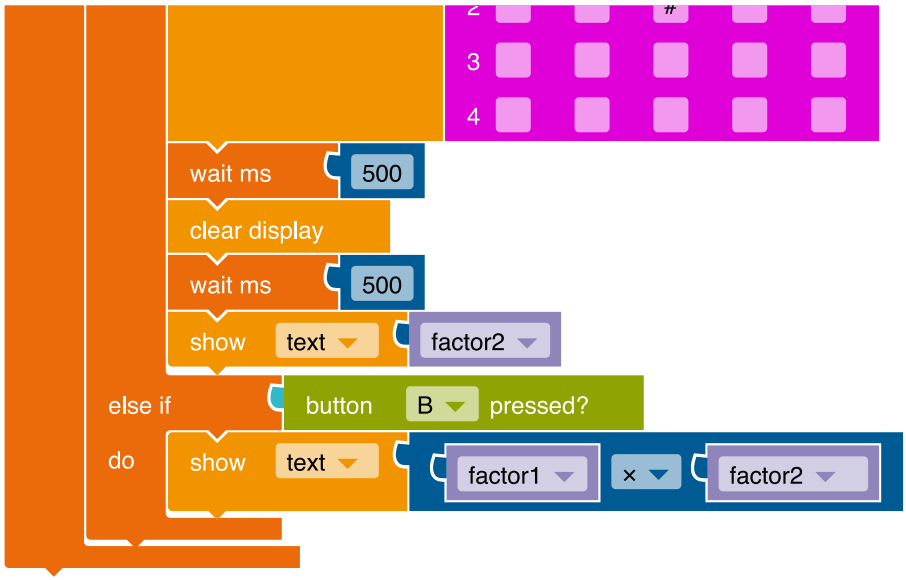
Now the two variables (here: factors) have to be inserted: Select from the category **Variables** the blocks *factor1* and *factor2* . Click on the "+" sign and select the multiplication sign "×" from the selection menu.



7 As in Code 1, the multiplication problem is not easy to read at first. Insert **pauses** at the appropriate places and/or clear the LED screen:

Control ‣ Wait
‣ "Wait ms"

Action ‣ Display
‣ "Clear screen".



8 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

- This code works well for working in pairs. The students can question each other or practice at home and on the go with their self-programmed mental arithmetic trainer.
- Note on handling: If you have selected the Touch-Pins as sensors, proceed as follows when executing with the *Calliope mini*: Hold the minus pin with one hand and touch the relevant pin with a finger of the other hand.

Expansion possibilities

- Programming further arithmetic operations:
  A modification of the addition trainer program is ideal for use in primary schools. Subtraction and division are a bit more complicated, since, among other things, the result may have to be displayed as a minus or decimal number.
- Send arithmetic problems:
  The category **Messages** offers the possibility to let several *Calliope mini* microcontrollers communicate with each other. Calculation tasks can be sent from *Calliope mini* to *Calliope mini* or to different boards at the same time.

Determine neighbouring numbers with the Calliope mini

The exercise

The *Calliope mini* is programmed to be a computer which, when requested, outputs a random number and displays the two neighbouring numbers (predecessor and successor) by pressing a button.

Subject matter

Mathematics

The students consider which arithmetic rule should be used to determine neighbouring numbers and how this task can be implemented using programming. With the self-programmed computer you can finally practice the determination of the predecessor and successor of random numbers controlling yourself.

Position in the syllabus

Content-related skills

In the field of *Numbers and operations*, the students can recognise, describe and represent number properties and number relationships as well as recognise, describe and continue laws in arithmetic patterns by correctly programming a *Calliope mini* neighbouring number trainer.

Process-related skills

In the field of *Reasoning*, the students can recognise and describe mathematical relationships and justify their own ways of thinking and finding solutions.

Requirements

Programming focus:

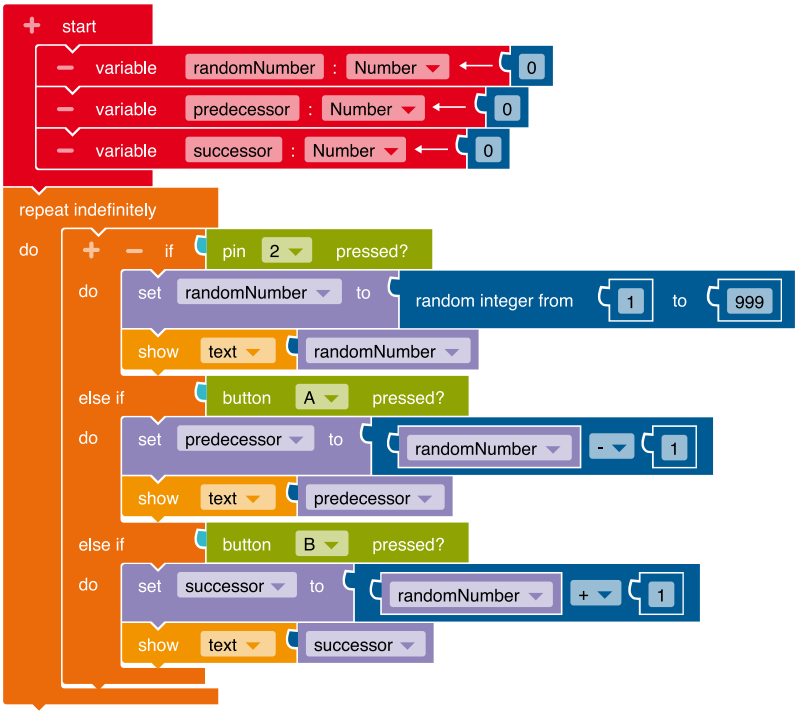
- Input via Touch-Pins and buttons, output via LED screen
- Structures: conditional statement, infinite loop, branch
- work with variables
- *NEPO*® categories used: Action, sensors, controls, logic, mathematics, variables

Programming difficulty:

- Medium level
- *NEPO*® Level: Expert

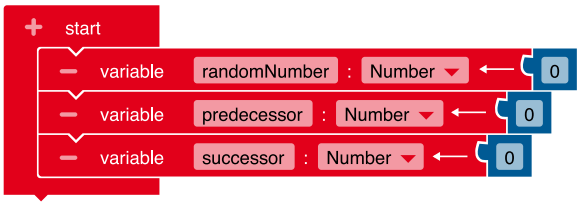
The code

This is what the code of the finished program looks like. This is defined step by step below.

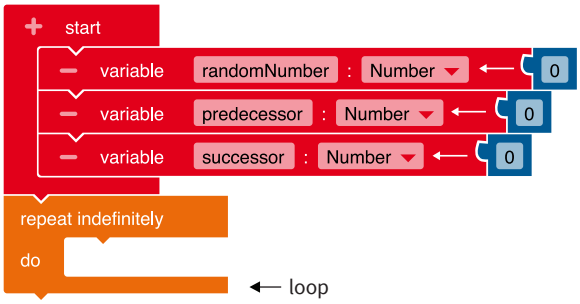


Steps to create the program for the output of neighbouring numbers

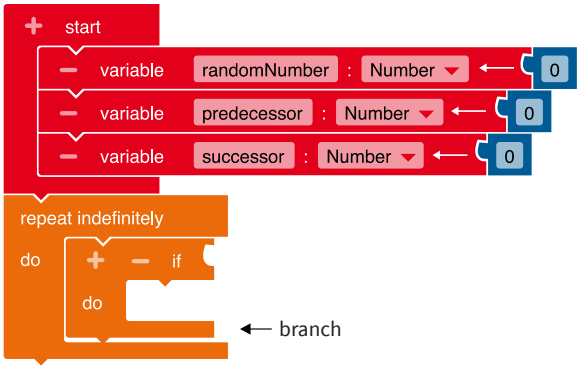
1 So that new numbers can be output again and again randomly when practicing with the neighbouring number trainer, various variables must first be created and defined: To create a new **variable**, click on the "+" to the left of "Start". A new block appears. Define the name (e.g. *randomNumber*, *predecessor*, *successor*) on the left and the data type (number) of the variable on the right.



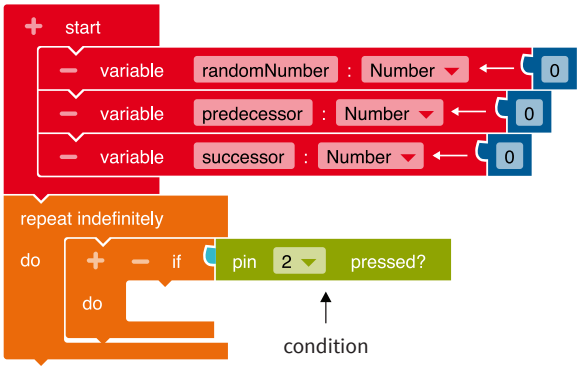
2 Unlimited random numbers should be able to be output: For this you need an infinite **loop**. Select the block "repeat indefinitely/do" from the category Control › Loops and add it to the "Start" block.



3 When Pin 2 is pressed (*if*), a random number should be output (*do*):
To establish this **condition**, you need a **branch**. Select the block "if/do" from the category Control › Decisions and insert it into the loop.

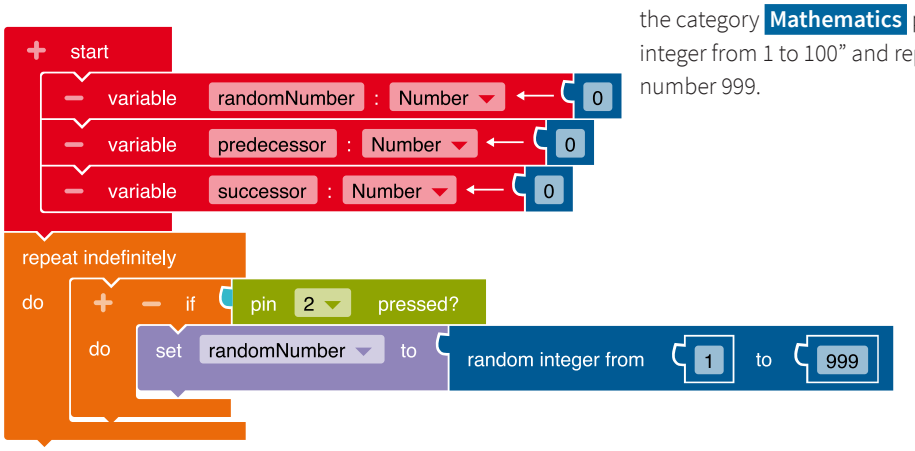


4 The desired input option (here via the sensor "Pin 2") is selected:
Select the "Pin 2 pressed?" block from the category Sensors and add it to the branch as a condition (blue area).



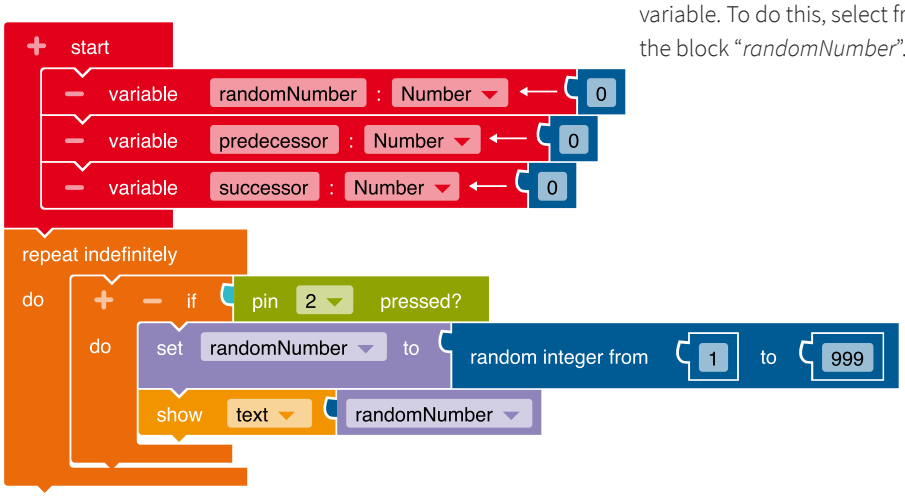
5 The value of the variable *randomNumber* must be defined:

Select from the category **Variables** the block "Write *randomNumber*" and insert it into the branch. Then from the category **Mathematics** place the block "random integer from 1 to 100" and replace the value 100 with the number 999.



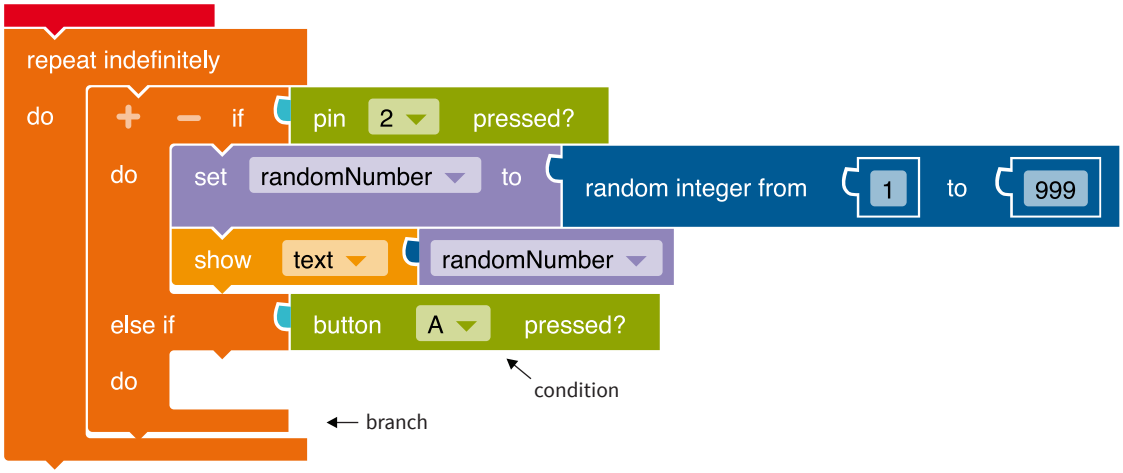
6 The random number should be displayed on the LED screen:

To do this, select the block "Show text" from the category **Action › Display** and replace the text "Hello" with the variable. To do this, select from the category **Variables** the block "*randomNumber*".



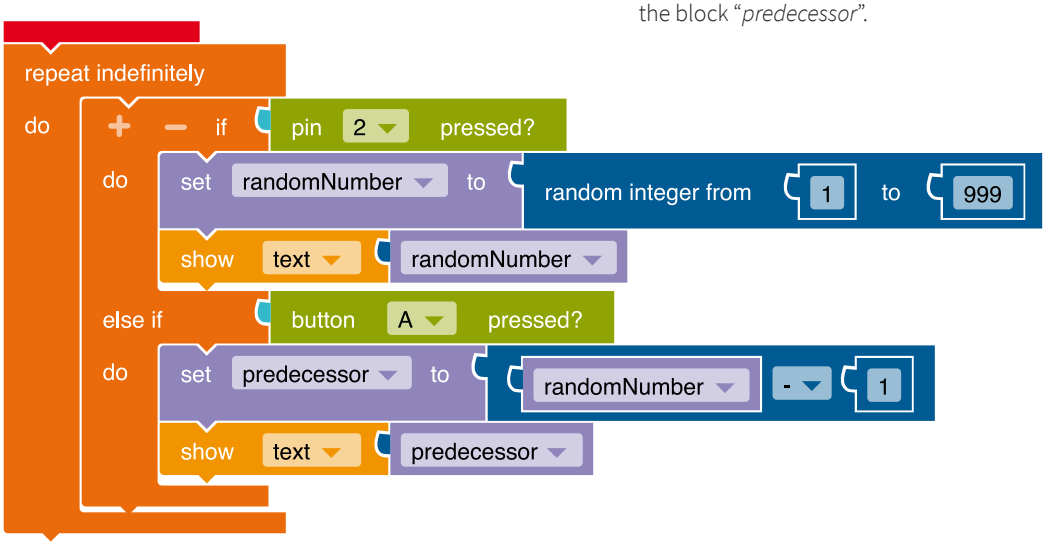
7 The predecessor of the output random number should be determined:
Another branch is needed for this.
In the branch, click on the "+" next to the "if".

The desired input option (here via the sensor "Button A") is selected:
Select the block "Button A pressed?" from the category **Sensors** and add it as a condition to the branch.



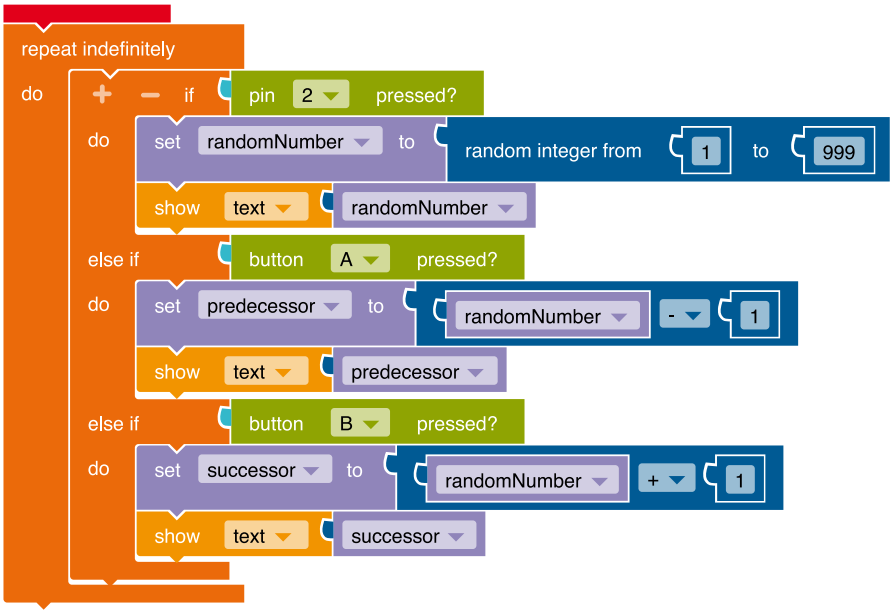
8 The value of the variable *predecessor* must be defined (randomNumber - 1 = predecessor):
Select the block "write predecessors" from the category **Variables** and place the block from the category **Mathematics** on it.
Select from the category **Variables** the block "*randomNumber*" and drag the variable into the left gap of the mathematics block.

Select the block "0" from the category **Mathematics** and insert it in the right gap. Replace the zero here with a 1. Click on the "=" sign and select the subtraction sign "-" from the **drop-down menu**.
The predecessor should be displayed on the LED screen: To do this, select the block "Show text" from the category **Action › Display** and replace the text "Hello" with the variable. To do this, select from the category **Variables** the block "*predecessor*".



9 The successor of the output random number should be determined and displayed on the LED screen (randomNumber + 1 = successor):

Proceed as in step 8. Just replace the variables (*predecessor* with *successor*) and select the addition symbol instead of the subtraction symbol in the selection menu.



12 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

In addition to practicing with controlling themselves, the students can also question each other in pairs.

Expansion possibilities

- Insert further summands and minuends (e.g. + 3, - 3, + 10, etc.).
- Doubling and halving (× 2, ÷ 2)

The Calliope mini and the Nim game

The exercise

In this programming example, the *Calliope mini* is to implement a simple mathematical game, the Nim game. In the Nim game, one, two or three elements are alternately removed from an initial set (in the original game, sticks, stones or pieces). The player who takes the last element wins. The game is programmed for two players, with the *Calliope mini* taking over the role of the second player. The second player (i.e. the *Calliope mini*) does not act tactically, but takes away a random number (one, two or three) of elements. The remaining number of elements is displayed on the LED screen after each turn. When an input from the first player is expected, the RGB-LED lights up yellow, if it is the *Calliope mini*'s turn, the RGB-LED lights up blue. The number of elements to be removed is determined by the first player by repeatedly pressing Button A. Pressing Button B concludes a move.
If the *Calliope mini* has won and the human player has lost, a sad smiley is displayed. However, if the human player wins, a smiling smiley appears on the LED screen.

Subject matter

Mathematics

With the Nim game, the students quickly notice that it is not a game of chance. By carefully observing, describing and justifying the various game process, you can individually develop your winning strategy. In this way, the Nim game contributes to a basic mathematical understanding.

Position in the syllabus

Content-related skills

In the field of *Numbers and Operations*, the students can solve puzzles by trying (e.g. disordered and systematic trying) by playing the Nim game with the *Calliope mini* tactically skilfully.

Process-related skills

In the field of *Problem solving*, the students can develop solution strategies, use solution strategies (e.g. systematic experimentation) and recognise and use dependencies.

Requirements

Programming focus:

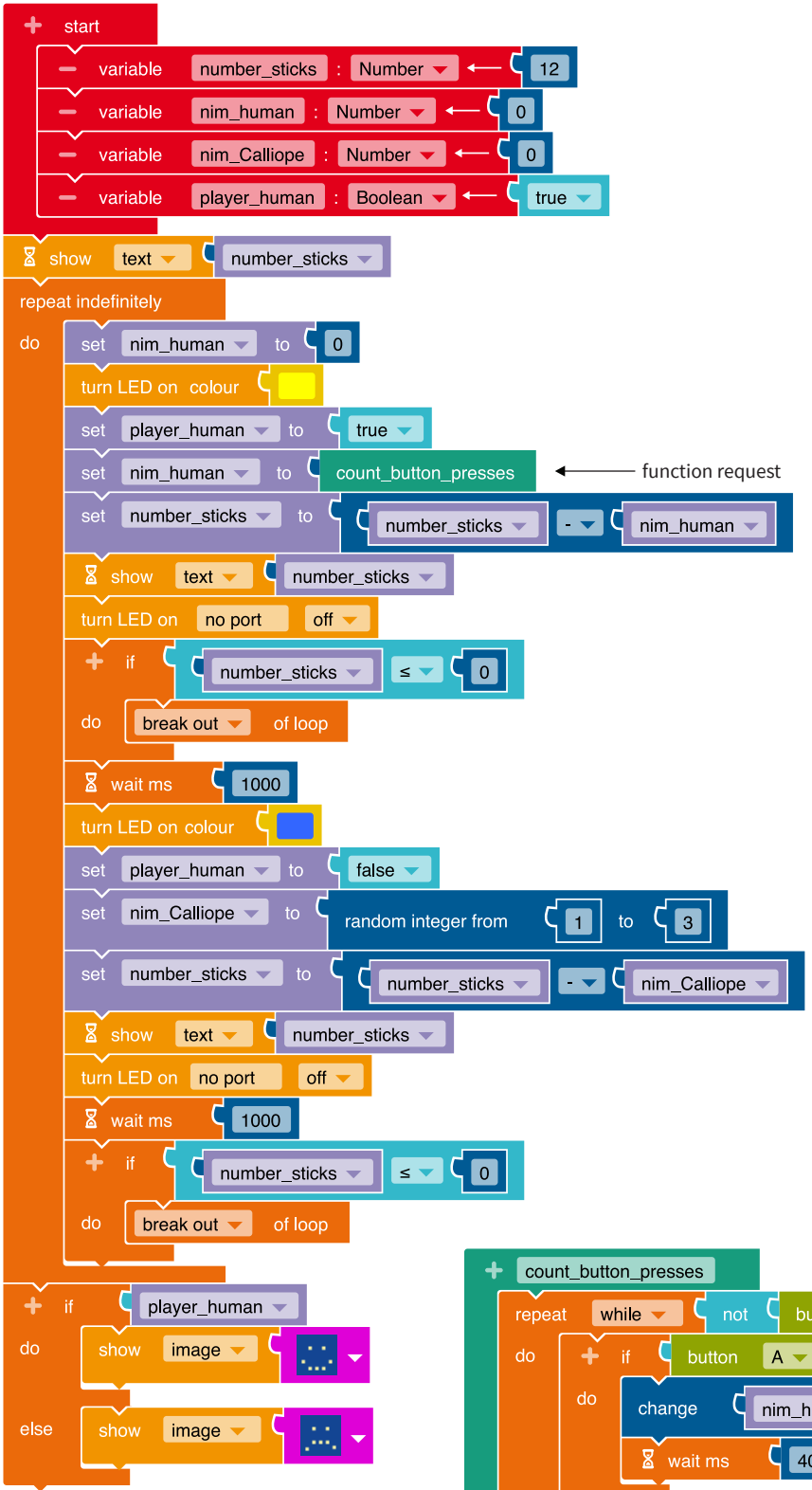
- Input via buttons, output via LED screen and RGB-LED
- Structures: conditional loops, infinite loops, branches and conditions
- deal with logical variables
- Functions (Count button presses)
- *NEPO*® categories used: Action, Sensors, controls, logic, mathematics, variables, functions

Programming difficulty:

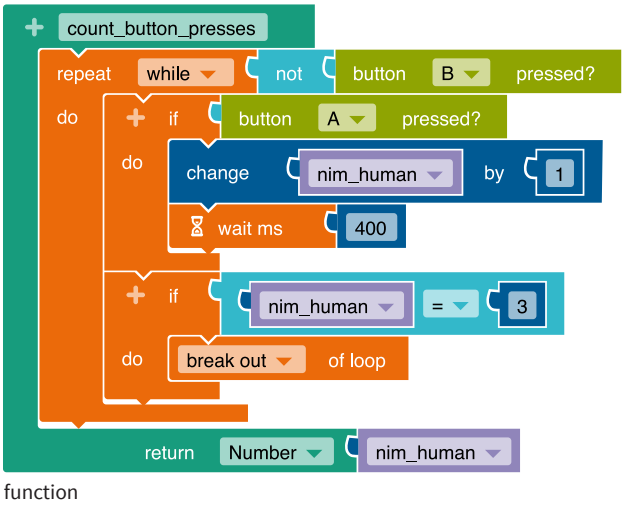
- Advanced level
- *NEPO*® Level: Expert

The code

This is what the code of the finished program looks like. It is defined step by step below.



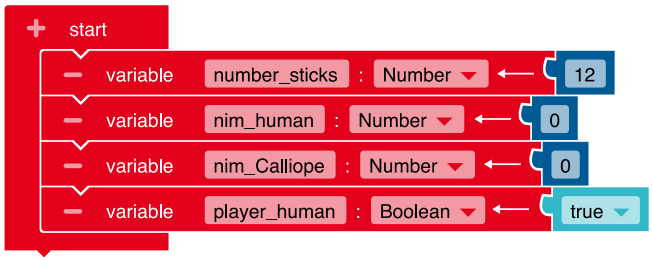
main program



function

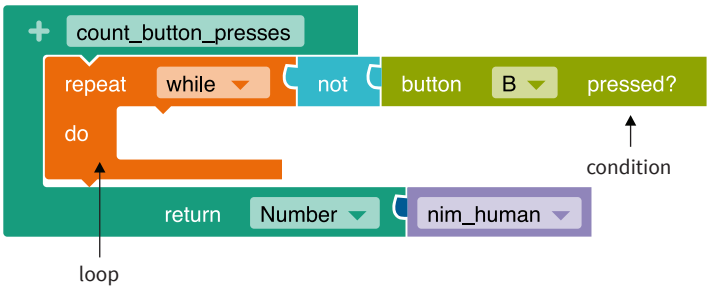
Steps to create the program for the Nim game

1 For the game process, the initial amount of sticks (a), the sticks taken away by the human player (b), the sticks taken from the program (c) and the current player (d) must each be saved in a **variable**:
For this purpose, four variables are created, three of which are of the **data type** "number" with the names
(a) "*number_sticks*". Click on the "0" in the blue field and enter the value "12".
(b) "*nim_human*"
(c) "*nim_Calliope*"
(d) You also need a variable "*player_human* " (d) of the data type "logical value". This can assume the two values *true* (it is the human's turn) or *false* (it is the *Calliope mini*'s turn).



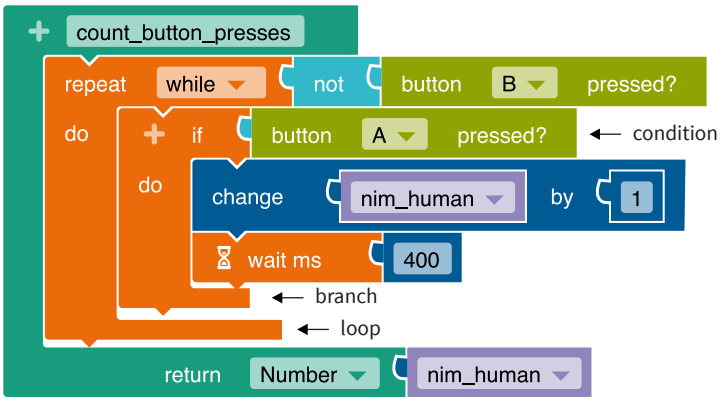
2 Since it is not so easy to count how often the human player has pressed the button A in a row this task is outsourced to a **function**. This keeps the main program clearer:
From the category **Functions** insert the block "doSomething ... return number" next to the main program on the free space and name this function *count_button_presses* (input via the keyboard).
This creates a new block with the name you just entered in the category **Functions**.
Add the block "repeat while/do" to this function from the category **Control › Loops**.
The **condition** of the **loop** is defined from the categories **Logic** with the block "not" and **Sensors** with the block "Button B pressed?".

function



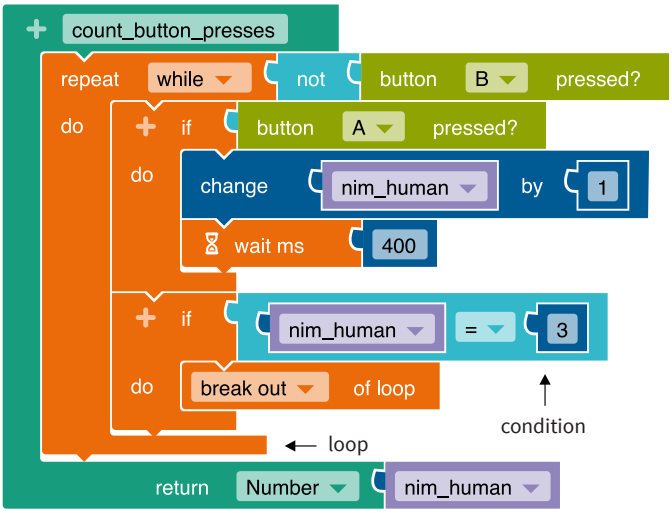
3 In this **loop** insert a **branch**. To do this, select from the category **Control › Decisions** the block "if/do" to which you dock the block "Button A pressed?" from the category **Sensors** as a condition.
The block change by is now inserted from the **Mathematics** category. The placeholders are filled with the block "*nim_human*" from the category **Variables** and the numerical value "1" from the category **Mathematics**. The button presses are thus counted.
From the category **Control › Wait** add the "Wait ms (400)" block.
This is used to prevent a long button press from counting as a double button press.
The function returns the value of the **variable** *nim_human* to the main program.

function



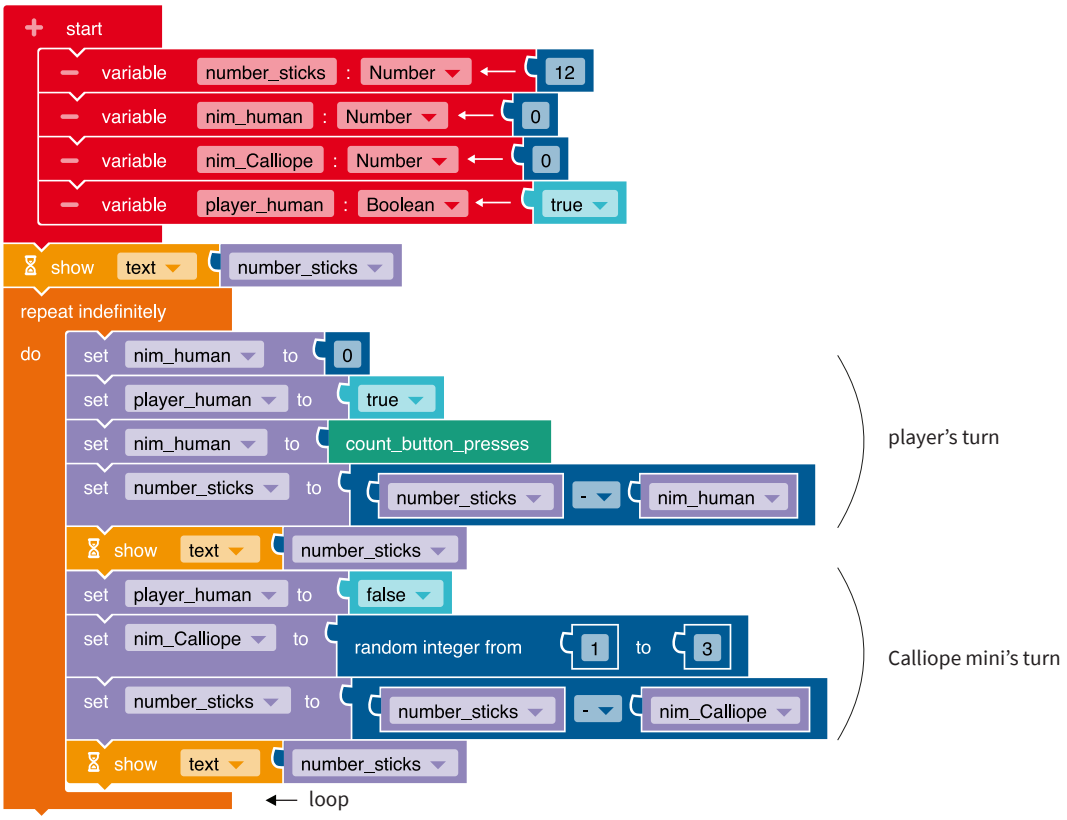
4 So that no more than three button presses are evaluated, a second branch from the category **Control ‣ Decisions** with the block “if/do” is inserted.
The condition consists of the equality block from the category **Logic**. The variable *nim_human* and the number “3” from the category **Mathematics** are used.
If this condition is fulfilled, the loop “repeat while Button B is not pressed?” is ended by the block “break out the loop” from the category **Control ‣ Loops** and the function is also exited.

function

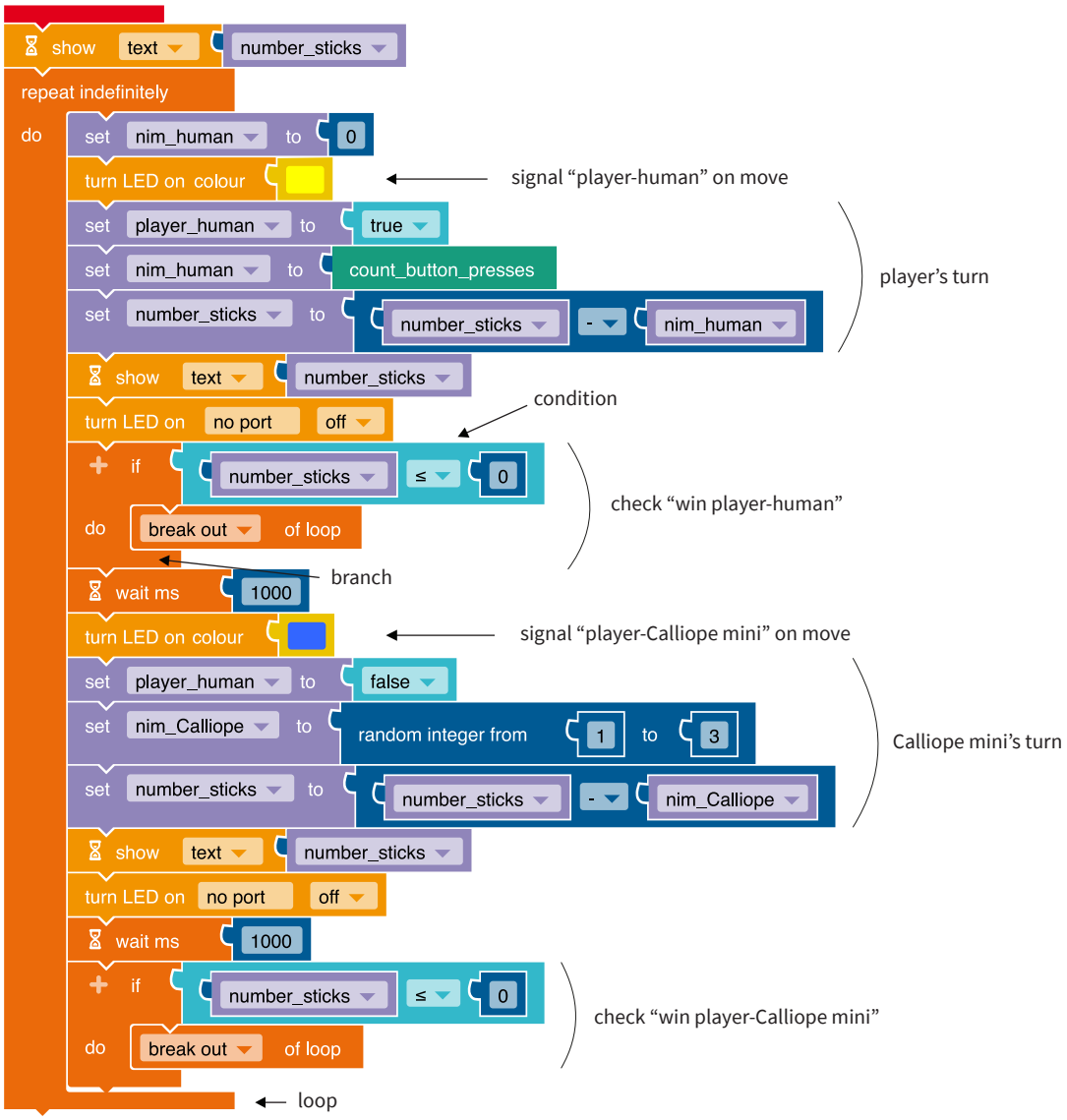


5 In the main program, the human player and the *Calliope mini* alternate. Therefore, the commands for the respective moves must follow one another:
The “Write” block is used seven times from the **Variables** category.
The variable *nim_human* must be reset to the value “0” before each move - a new move is due.
The **variable** *player_human* gets the value “true” from the category **Logic** – It is the human player’s turn and the **variable** *nim_human* gets the value of the function from the category **Function** *count_button presses* – no more than three sticks may be removed.

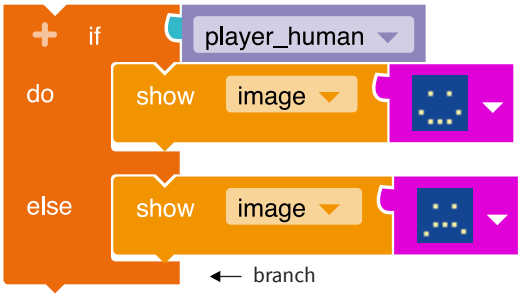
After the move is completed, the number of sticks is reduced accordingly by subtracting from the category **Mathematics**.
Do the same for the *Calliope mini*’s turn. The only difference is not to reset the variable *nim_Calliope* to “0”, since it is overwritten every time by the block “random integer from 1 to 3” from the category **Mathematics**.
At three points in the course of the game (at the beginning and after each move by one of the two players) the value of the variable *number_sticks* should appear as text on the LED screen. To do this, select the “Show text” block from the category **Action ‣ Display** and insert it as shown below.



6 The *Calliope mini* should determine and display who has won:
If the number of sticks equals zero, the player whose turn it is wins. The infinite **loop** is cancelled in this case.
After the human's move or the *Calliope mini*'s move insert the block "if/do" from the category **Control › Decisions**.
Insert the condition from the category **Logic** "≤" and compare the variable *number_sticks* with the numerical value "0" from the category **Mathematics**. You can find the block "break out loop" in the category **Control › Loops**.
It is helpful to assign a colour of the RGB LED to each of the players, which always lights up while it is their turn. To do this, you need a block for turning on and off from the category **Action › Light status**. As the *Calliope mini* calculates very quickly, you incorporate an artificial delay of one second (= 1000 milliseconds) from the category **Action › Wait** for each player. The number "1000" can be found in the category **Mathematics** in the block "0" and change the value to "1000".



7 At the end of the game, various symbols are displayed on the LED screen to show which player has won:
To do this, a branch from the category **Control › Decisions** with the block "if/do/else" is required, which checks the value of the **variables** *player_human* after the end of the game loop. If this value is true, the human player has won.
This is displayed by means of the image of a laughing smiley from the category **Action › Display** and the corresponding selection via the **drop-down menu**. Use a sad smiley in the event that it was the *Calliope mini*'s turn at the end of the game, i.e. the **variable** *player_human* has the value "false".



8 Transfer the program code to your *Calliope mini* by downloading the program and saving it on the *Calliope mini* (calliope.cc/en/lets-start/start-coding).

Notes and information

Implementation scenarios in class

- Finding winning strategies, for example the number of sticks to be taken in moves, that enable a win.
- Address the difference between chance (luck) and strategy.
- Questions for discussion: Is a computer intelligent? If it is not, why can it still slip into the role of a player?
- Compare the meaning of the rules of the game in different contexts (e.g. sports, board games).

Expansion possibilities

- Changing the rules of the game, for example only one or two sticks may be removed. Investigate what changes in the program or in the game strategy.
- Change of the number of sticks at the beginning of the game.
- The player simulated by the *Calliope mini* can be made more intelligent, for example by trying further branches to avoid certain numbers of sticks or to reach others.

Glossary

Instruction The blocks in the *NEPO*® editor that have a triangular notch at the top left and a triangular extension at the bottom left are instruction or command blocks.
The term “instruction” is synonymous.
Programs are made up of instruction sequences (sequences) and control structures (e.g. loops, branches).

Output All actions of the *Calliope mini* that can be heard or seen are understood as output. As output, sounds can be generated with the loudspeaker, images or texts can be displayed on the LED screen or the RGB LED can light up in a colour.
Invisible outputs of the *Calliope mini* are radio communications, motor control signals or electrical voltages at the pins.

Drop-down menu Some program blocks have a drop-down menu, for example in Sensors ▸ “Button A pressed?”. There, after selecting the block, you can also select the Button B by clicking on the selection area. The current selection is indicated by a tick.

Condition A comparison block from the category Logic is required to define a condition. Variables or constant values (e.g. a number, a text) must be inserted to the left and right of the comparison operator. The two compared values must be of the same data type. The result of a condition is a truth value (true or false). Conditions can be combined as desired using Logic ▸ “and/or” or negated using Logic ▸ “not”.

Command see instructions

Block A block is a graphic program element of the *NEPO*® editor. These blocks can be found in the categories. For example, they are selected and put together with the mouse.

Code A program in the *NEPO*® editor consists of at least one main program (beginning with “Start”) and any number of functions. The word “code” can be understood as the English language short form of the term “programming code” also known as “program”. Coding here means that each programming language has its own hard-coded commands.
A program has two display forms in the *NEPO*® editor: (1) The editable coloured program blocks and (2) the non-changeable source code (visible via the symbol at the top right in the editor window “<>”). This corresponds exactly to the program blocks.

Data type This is the range of values that a variable can have. There are defined operations for each data type, such as the arithmetic operations for the data type number.
Examples of elementary data types are: integers, letters, logical values.
Examples of compound data types are: Strings, lists.

Editor There are three editors for the *Calliope mini* (*Calliope mini* Editor, PXT, *NEPO*® in the *Open Roberta Lab*®). A program for the *Calliope mini* can be created or changed with the help of an editor. The program can either be created using graphic program blocks or be text-based.

Input Input is everything that can be processed by the sensors and buttons of the *Calliope mini* in a program.

Function If a sequence of instructions is required frequently, it makes sense to outsource it to a function. This function is given a clearly defined name and can be called up under this name as often as required in the main program. A new program block is created. A function can contain a return value and parameters.

Index The index denotes the position of a list element within a list. The first element receives the index zero, for each additional element the index is incremented by one. For example, if the indices range from 0 to 3, the list contains 4 elements.

Category The instructions, control structures, conditions, functions and variables are grouped into content categories in the *NEPO*® editor. The different categories are shown in different colours. The corresponding blocks are each the same colour.

LED screen
The 25 red light-emitting diodes on the front of the *Calliope mini*, which are arranged in a square in five rows and five columns, are called the LED screen. Simple pictures as well as letters and numbers as well as text can be displayed on it as a still image (Category Action ‣ Display).

List
If several variables of the same data type and meaning are used in a program, we recommend the sequential data type list, which has any number of containers of a data type to be specified (number, logical value, string). In the *NEPO*® editor, three list elements are initially displayed. By clicking on the "+" or "-" symbols, their number can be increased or decreased.
There are special functions for lists, such as searching for a list item.

Pauses
As the *Calliope mini*, like any computer, can calculate and execute commands much faster than a human, it sometimes makes sense to stop it for a certain period of time in order to visualise the intermediate status of the program. For this purpose, appropriate blocks can be selected in the *NEPO*® editor in the category Control ‣ Wait.

Pin
Pins refer to the six corners of the *Calliope mini*. The two upper corners ("+" and "-") allow the connection of a voltage source (battery), the four lower corners ("P0" to "P3") are touch-sensitive (Touch-Pins) (see example "Mini piano"). External sensors can also be connected to P0 to P3.

Program
see Code

RGB-LED
An RGB LED is a light emitting diode (LED) that can light up in several colours.
At its core there are three light-emitting diodes whose colour components (R-red, G-green, B-blue) are mixed additively. The colour white is created by the simultaneous lighting of all three colour components (Category Action ‣ Light status).

Loop (infinite/ conditional/ counting)
A loop allows a sequence of statements to be executed repeatedly. The loop condition must be true) for the statement sequence to be executed. If it is not fulfilled, the program sequence is continued after the loop.
An *infinite loop* has no condition, but instead the constant logical value "true".
A *counting loop* contains, instead of a condition, a counting rule that specifies how often the loop is runthrough.

Sensor
The *Calliope mini* has various electronic sensors that can determine the properties of the environment (category Sensors).
The orientation in the earth's magnetic field (compass sensor), the temperature, the brightness, the position of the *Calliope mini* in the room, noises (microphone), the touch of the four pins as well as the buttons A and B can be determined.

Simulation
If there is no *Calliope mini* at hand, a program written in the *NEPO*® editor can be simulated by clicking on the "SIM" button at the top right of the window. A window with a *Calliope mini* opens and the program can be started using a "Play" button at the bottom of the window.

Position
see Index

Button
In contrast to a switch, a button jumps back to its original position when it is pressed. A button therefore has exactly two states: pressed and not pressed. The *Calliope mini* has buttons A (blue) and B (red), which can be queried in programs, as well as the reset button (white).

Touch-Pin
see Pin

Variable
A variable is a container for a certain value that is defined at the beginning (click on "+" in the start block) and can later be changed as often as you like. The value is changed via an assignment in the program. The value of a variable is saved for the duration of the program execution.
Each variable needs a unique name that must begin with a letter. The values of a variable belong to a data type to be specified in the variable setting (number, string, logical value, etc.).

Comparison
A comparison is usually part of a condition. The current values of two variables can be compared with one another. In addition to equality, there are the comparison operators "not equal to", "greater than", "less than", "greater than or equal to" and "less than or equal to".

Branch Depending on the truth value of a condition (true or false), the linear sequence of a program is branched into two different program sections (Category Control › Decisions). The program sequence changes depending on which truth value is determined by the *Calliope mini*. Any number of instructions or other control structures can be used in each of the two branches. After a branch, the program continues linearly.

Truth value see Condition and Branch

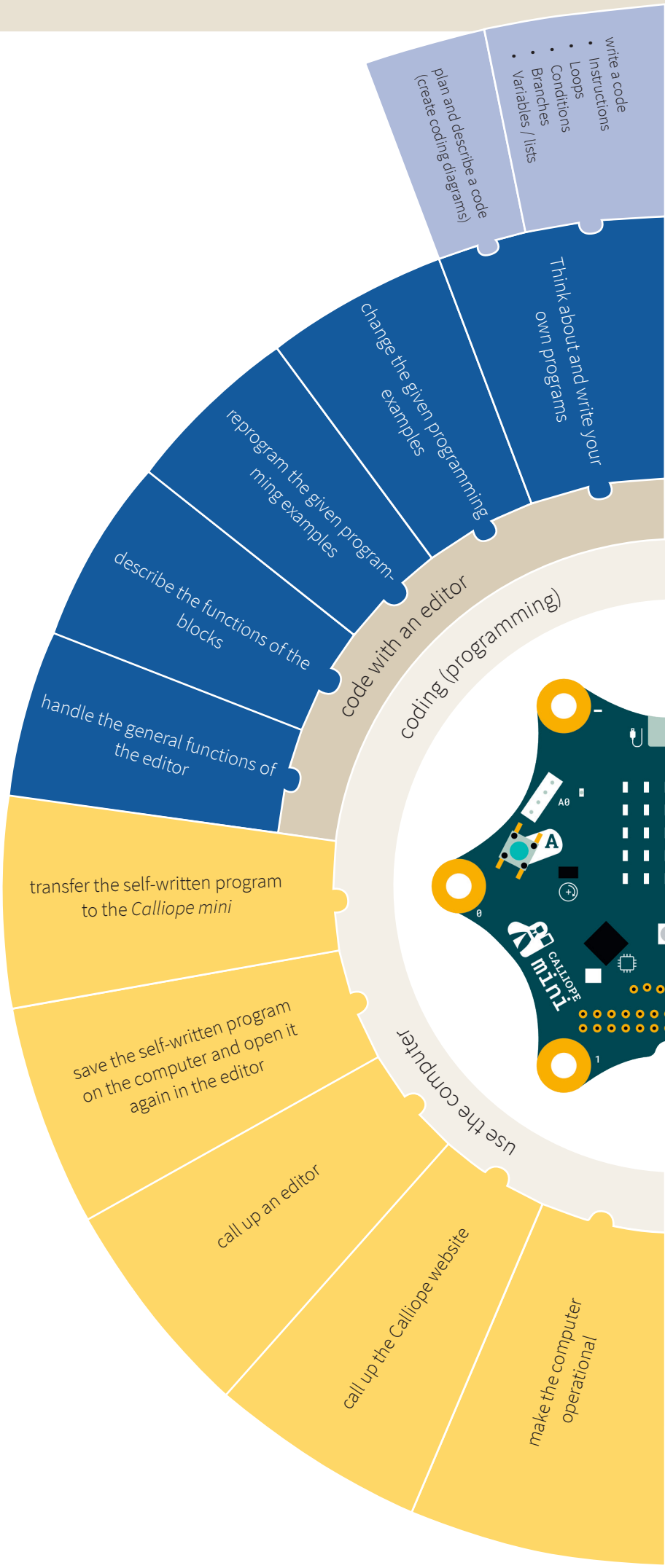
Wait see Pauses

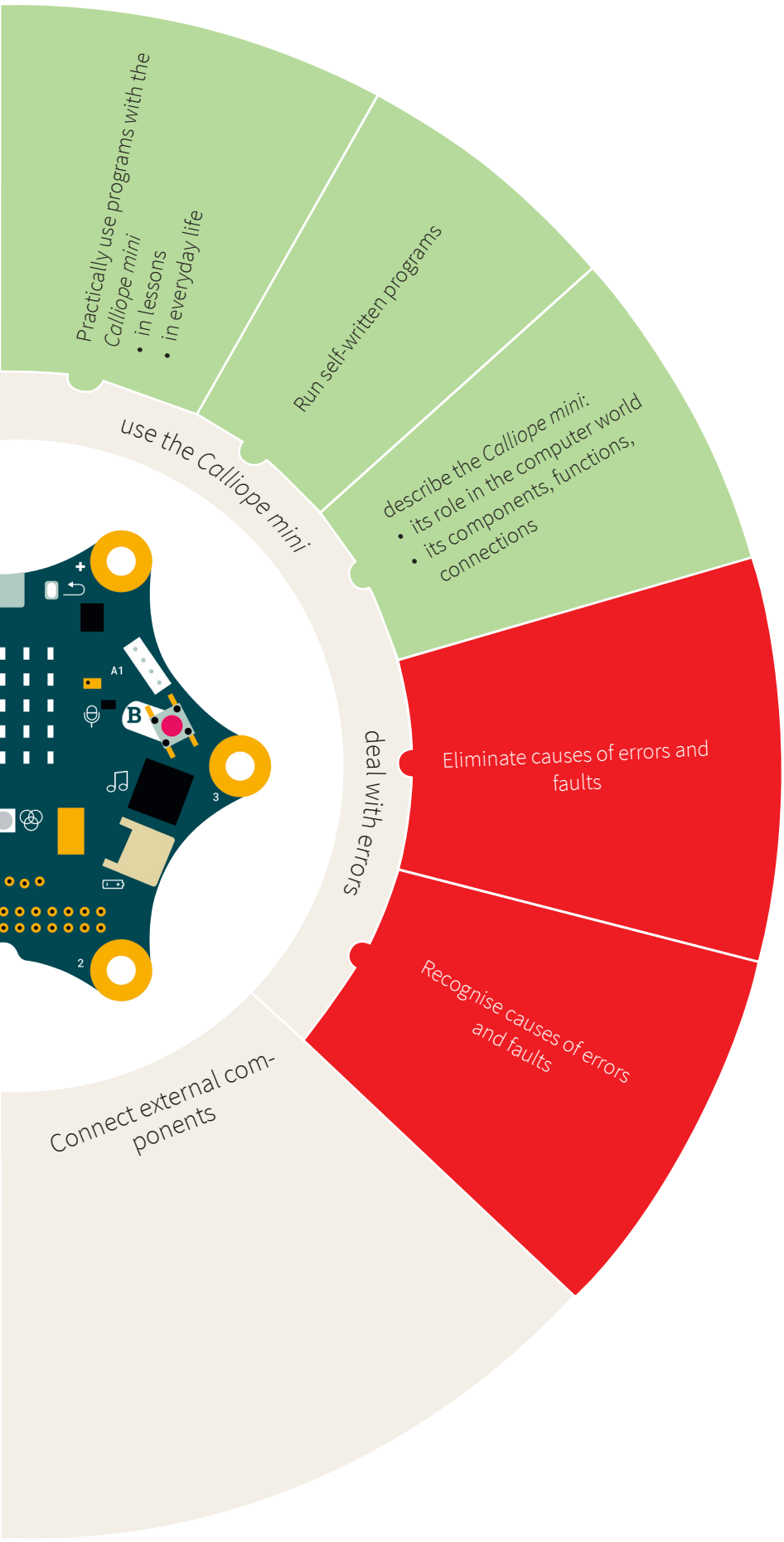
String A string is a sequence of letters, punctuation marks or special characters. In the *NEPO*® editor, it is also referred to as text and placed in quotation marks. Variables can, among other things, belong to the data type “string”.

Timer The processor starts a timer at the same time as the program transferred to it, which works like a stopwatch with an accuracy of milliseconds (thousandths of a second). The current status of this timer can be queried any number of times using the block “get value ms timer 1” in the category Sensors. It is also possible to start this timer again with the “reset ms timer 1” block.

Random A random numerical value can be generated by the *Calliope mini* processor. A range of numbers within which the random number lies can be defined by the programmer.

This learning map includes all the important skills that are required to use the computer and an editor independently and to program the *Calliope mini*. It was developed on the basis of diverse practical experience. As the official guidelines and syllabus do not (yet) provide a binding syllabus for coding, the learning map cannot claim that the learners have fully mastered all skills at the end of a series of lessons. Rather, it should be used in the form of an overview and step-by-step guide as a didactic guideline for a larger coding lesson project.





The learning map is divided into the three elementary main areas *using the computer*, *code* and *using the Calliope mini* and is supplemented by the two areas *dealing with errors* and *connecting external components*. Within the main areas, the linear increase in competence becomes clear, readable from bottom to top. The competence *connecting external components* offers many possibilities to use the *Calliope mini* in practice, but is not detailed because it has not yet been specified in this textbook. The arrangement of the learning map as a pie chart makes it clear that the acquisition of skills is not only structured linearly, but is also based on a lively alternation of the individual skills.

In the present teacher material you are addressed as a teacher without showing you options for transferring the presented material into the classroom. We have compiled experiences and recommendations from feedback from teachers who have already started coding in class:

- Familiarise yourself with the structure and the possibilities of *NEPO*® and the *Calliope mini*.
- Try out the coding examples yourself.
- Use the free internet access to the editor for this. You can also practise programming with *NEPO*® without the *Calliope mini* and simulate your code.
- Get used to storing the codes on your computer in order to archive your codes and practice transferring the code to the *Calliope mini*.
- Vary existing codes according to your own creative ideas.
- Write your own (small) programs.
- Look for colleagues in the college and become creative together.
- Trying it out yourself not only serves to develop your creativity, but also to develop confidence if your students have questions and mistakes in class.

We hope that this spark has got across to you and that you want to try coding in your class. An indispensable requirement is the provision of the necessary digital devices for programming.

In the following, teachers describe their first steps and conditions for coding to be successful in everyday school life in primary school:

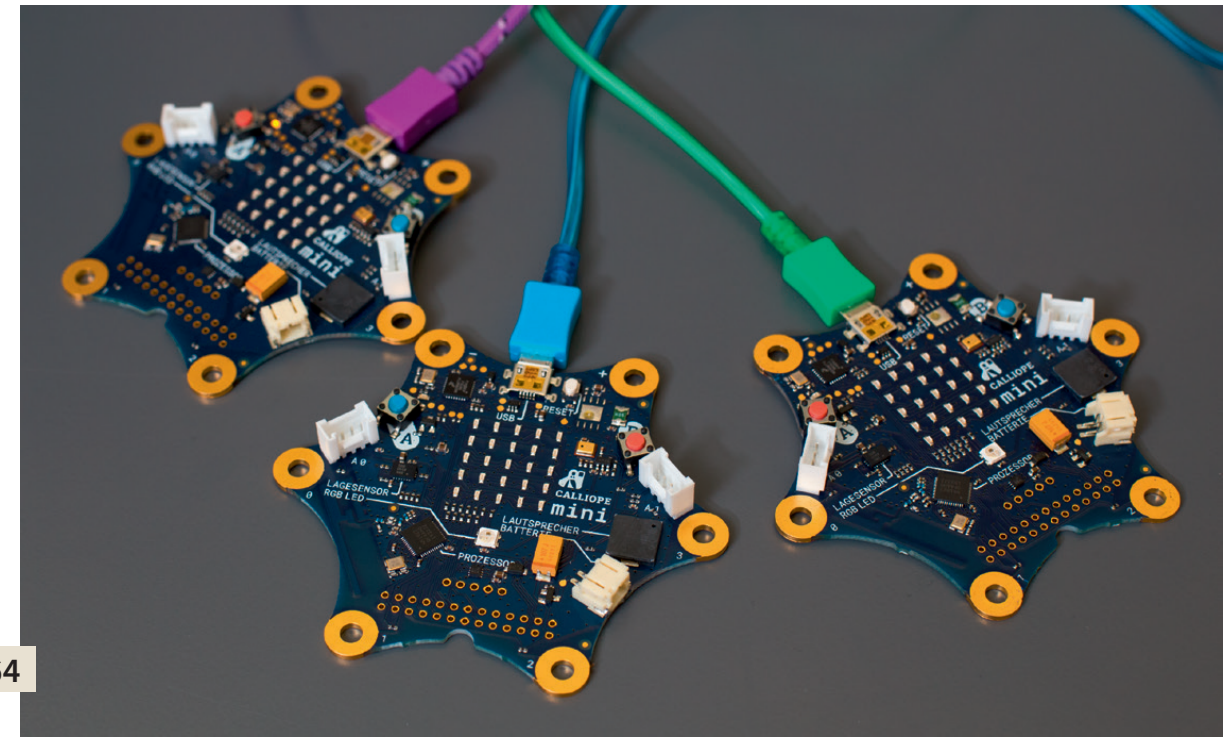
- two students per device (e.g. laptop, PC) for coding
- a *Calliope mini* for each student
- Entry with a smaller number of students, for example as part of an afternoon group with 10–12 children on 5–6 computers. An adult helper at the beginning of the coding course would be ideal to overcome the entry hurdles for the children (e.g. lack of basic computer knowledge, making mistakes, lack of routine in handling the blocks, ...).
- Encourage your students, in consultation with their parents, to use the free Internet access to the editor from home. The children can also practice programming without the *Calliope mini*. The *NEPO*® editor offers a good simulation of the written programs.

Your feedback is important to us!

Have you tried the examples at home or maybe even with your class? Share your experiences, ideas and tips with us. In this way you help to develop the materials in an optimum way with regard to the requirements in the classroom. We look forward to receiving your feedback! Write to us at calliope@cornelsen.de

Reference to student material

- Can be used from grade 3 onwards for the subjects English, general knowledge and mathematics
- based on the examples in the present teacher material



Cornelsen/Sibylle Baier

CALLIOPE

Calliope mini is a product of Calliope gGmbH

With the microcontroller ***Calliope mini*** every schoolchild in Germany from the 3rd grade onwards should be able to have playful access to the digital world. It is only if we have digital knowledge that we can all actively participate in society and help shape it.

For this purpose, experts from the IT and education sectors work together in an interdisciplinary manner as part of the Calliope team.

Find more information about the initiative at calliope.cc

OPEN Roberta®

The *Open Roberta Lab* is a freely available programming platform on which children, young people and adults can learn to program even without any previous knowledge. Schoolchildren intuitively bring the *Calliope mini* to life with the graphic programming language *NEPO*®. Open Roberta® is a technological advancement of the Fraunhofer initiative "*Roberta*® - Learning with Robots", which has been promoting digital education in Germany since 2002. The *Open Roberta* project was initiated by Fraunhofer IAIS in cooperation with Google. *Roberta*, *Open Roberta* and *NEPO* are registered trademarks of the Fraunhofer Society for Applied Research e. V.

Click here for the *Open Roberta Lab*: lab.open-roberta.org

Coding with the Calliope mini

The teacher materials for coding with the **Calliope mini - Programming in primary school**, directed towards teachers from grade 3.

- 11 coding examples with content from the syllabus of the subjects general knowledge, German and mathematics in grades 3 and 4.
- As a coding novice, you will program your first own programs step by step.
- You systematically build your coding skills using the content examples.
- Experience the Calliope mini as a fascinating tool for your school lessons.

Try it out and see for yourself how amazing and easy coding is!

CALLIOPE.CC

CALLIOPE