

An introduction to

SCRATCH

Table of contents

About this guide	4
1 Install Scratch	5
1.1 <i>Using scratch for the first time</i>	6
1.2 <i>Question space</i>	7
2 Introduction into Scratch	8
2.1 <i>Header</i>	8
2.2 <i>Block Palette</i>	9
2.2.1 <i>Block Shapes</i>	9
2.2.1.1 <i>Hat Blocks</i>	9
2.2.1.2 <i>Stack Blocks</i>	9
2.2.1.3 <i>Boolean Blocks</i>	9
2.2.1.4 <i>Reporter Blocks</i>	9
2.2.1.5 <i>C Blocks</i>	10
2.2.1.6 <i>Cap Blocks</i>	10
2.3 <i>Paint Editor</i>	10
2.4 <i>Sound Editor</i>	12
2.5 <i>Sprite Pane</i>	13
2.6 <i>Stage</i>	13
2.7 <i>Code Area</i>	14
2.8 <i>Quiz</i>	15
2.9 <i>Question space</i>	16
3 Motion	17
3.1 <i>Move and turn around!</i>	17
3.2 <i>Coordinates</i>	18
3.3 <i>Where to point</i>	19
3.4 <i>Exercises</i>	19
3.5 <i>Question space</i>	20
4 Operators	21
4.1 <i>Calculations</i>	21
4.2 <i>Comparison</i>	22
4.3 <i>String operations</i>	23
4.4 <i>Exercises</i>	23
4.5 <i>Question space</i>	24
5 Events & Controls	25
5.1 <i>Events on actions</i>	25
5.2 <i>Handling Messages</i>	26
5.3 <i>Loops</i>	27
5.4 <i>Conditionals</i>	27
5.5 <i>Wait & stop</i>	28
5.6 <i>Exercises</i>	29
5.7 <i>Advanced Exercises</i>	29
5.8 <i>Question Space</i>	30
6 Variables & Lists	31
6.1 <i>Manage a variable</i>	31
6.2 <i>Manage a List</i>	32
6.3 <i>Exercises</i>	34
6.4 <i>Advanced Exercises</i>	34
6.5 <i>Question Space</i>	34
7 Looks	35
7.1 <i>Talk to the player</i>	35

7.2 <i>Change your look</i>	35
7.3 <i>Move to spotlight</i>	37
7.4 <i>Exercises</i>	37
7.5 <i>Question space</i>	37
8 Sound	39
8.1 <i>Control your sounds</i>	39
8.2 <i>Exercises</i>	40
8.3 <i>Question space</i>	40
9 Sensing	41
9.1 <i>Touching something?</i>	41
9.2 <i>Ask the player</i>	41
9.3 <i>Key pressed and mouse movement</i>	41
9.4 <i>Loudness of the environment</i>	42
9.5 <i>Control the time</i>	42
9.6 <i>Get values of other objects</i>	42
9.7 <i>Exercises</i>	42
9.8 <i>Question space</i>	43
Appendix I – Cheat Sheet	44
Sources	45

About this guide

Welcome aboard your self-study scratch tutorial.


This guide is designed for people who have no experience in programming and want to get in touch with this art. If you are a curious person who always wants to learn something new. This is the right guide for you.

This guide is created as a self-study guide what takes about 12 hours to complete. You can follow the instructions in your own speed and at any time you want. All exercises do have solutions to compare, but do not check the solutions before you managed to get a working example. If your solution is not equal to the authors solution, don't worry. The most amazing part about programming is that there are thousands of solutions for a single problem. If your solution does solve the problem your solution is perfect! If you are not sure about your solution or have a question do not hesitate to contact the teacher. The teachers contact details can be found below.

As preparation for the session every section in the guide has a separate space at the end to write down all your questions you want to ask in the Session. Plan enough time to complete this guide and write as many questions as possible that we have an instructive session together.

All solution files and the file for the coding-program are combined inside a package. If you do not have downloaded the whole package yet, do it now by following this link <https://www.sven-waser.ch/scratch> or scan the QR-Code on the right.



Every exercise is marked with this  symbol. The filename of the solution in the package starts with the chapter number followed by an underscore and a two-digit exercise number. For example, the solution for the first exercise in chapter 1 has the filename "1_01.sb3".

As a student I know that it is hard to remember everything explained in the previous chapters. Therefore, I created the Appendix I what provides a Cheat Sheet containing the most important functionalities of the scratch language. Using this page while solving the given tasks and during the session will make your life a lot easier and saves a lot of time.

Happy Coding!

Teachers contact details:

Name: Sven Waser

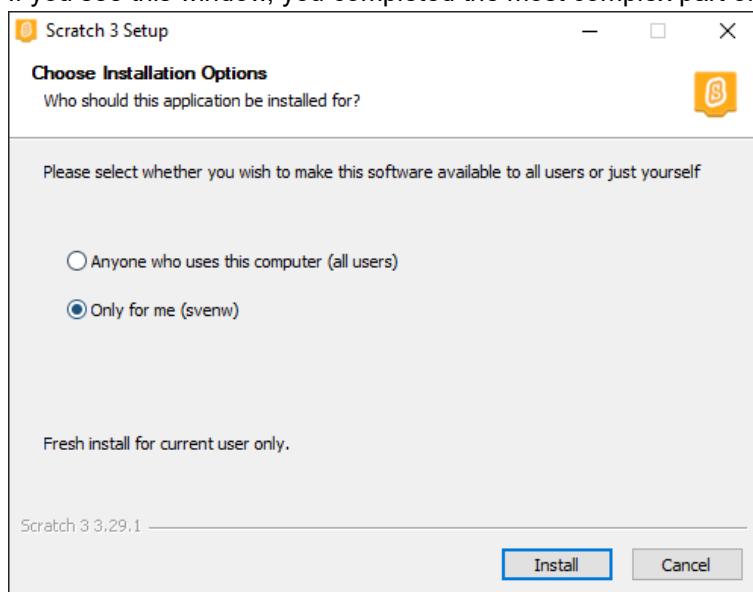
E-Mail: sven.waser@istep.ch

Editors note: This tutorial was written in spring 2024 for the iStep association. All code files were created with the Scratch Desktop software version 3.29.1 for Windows.

1 Install Scratch

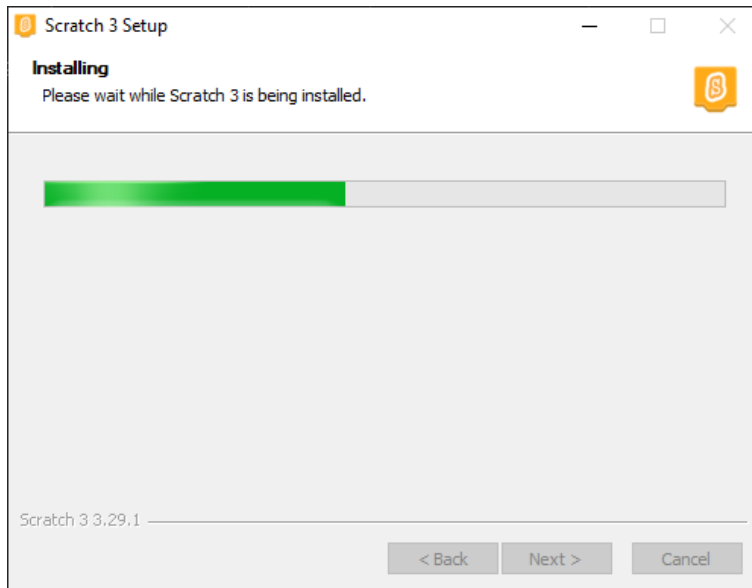
Let's start with scratch. Before we can start, we need to install the scratch program. If you have already installed scratch on your computer, you can skip this chapter and continue with chapter 2 Introduction into Scratch. If you are not able to install scratch, you can use the online version of scratch available on <https://scratch.mit.edu/projects/editor/> and start with chapter 2 Introduction into Scratch, else follow the instructions. You have never installed any program before. Don't worry it is not that complicated and there's always a first time. This guide will help you go through all necessary steps.

1. At the beginning open the tutorial package folder. If you have not yet downloaded the package folder, you will find a link and the QR-Code to scan in the previous chapter.
2. Search the file "Scratch 3.29.1 Setup.exe" and double click it what will open the installation dialog box.
3. If you see this window, you completed the most complex part of the installation.



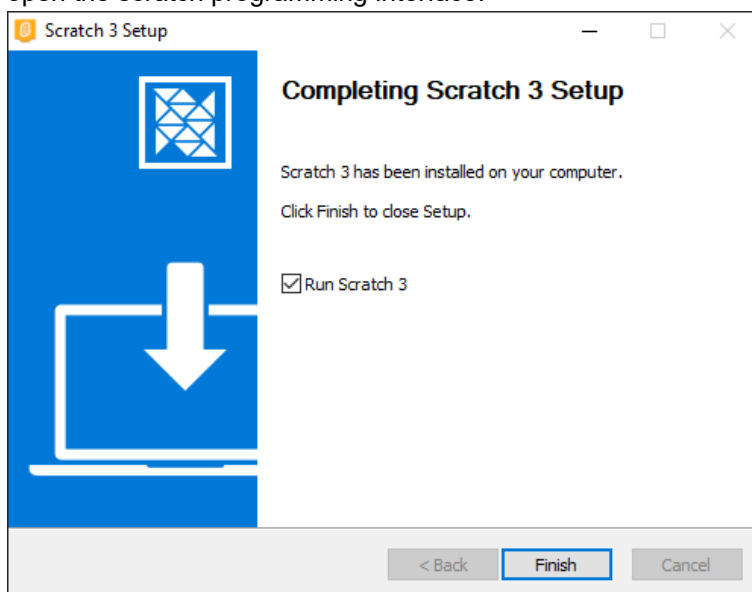
Installation window asking for installation location.

4. By default, the radio button should select "Only for me" otherwise select it. The information inside the brackets represents your username what will be different for anyone doing this tutorial. In this case the username is svenw.
5. After selecting the right option click on install and the installation window with the progress bar should be visible to you.



Installation window installing scratch.

6. Wait until the progress bar is completed and click the next button as soon as it is available.
7. You have done it. The final screen appears. Click on finish to complete the installation and to open the scratch programming interface.

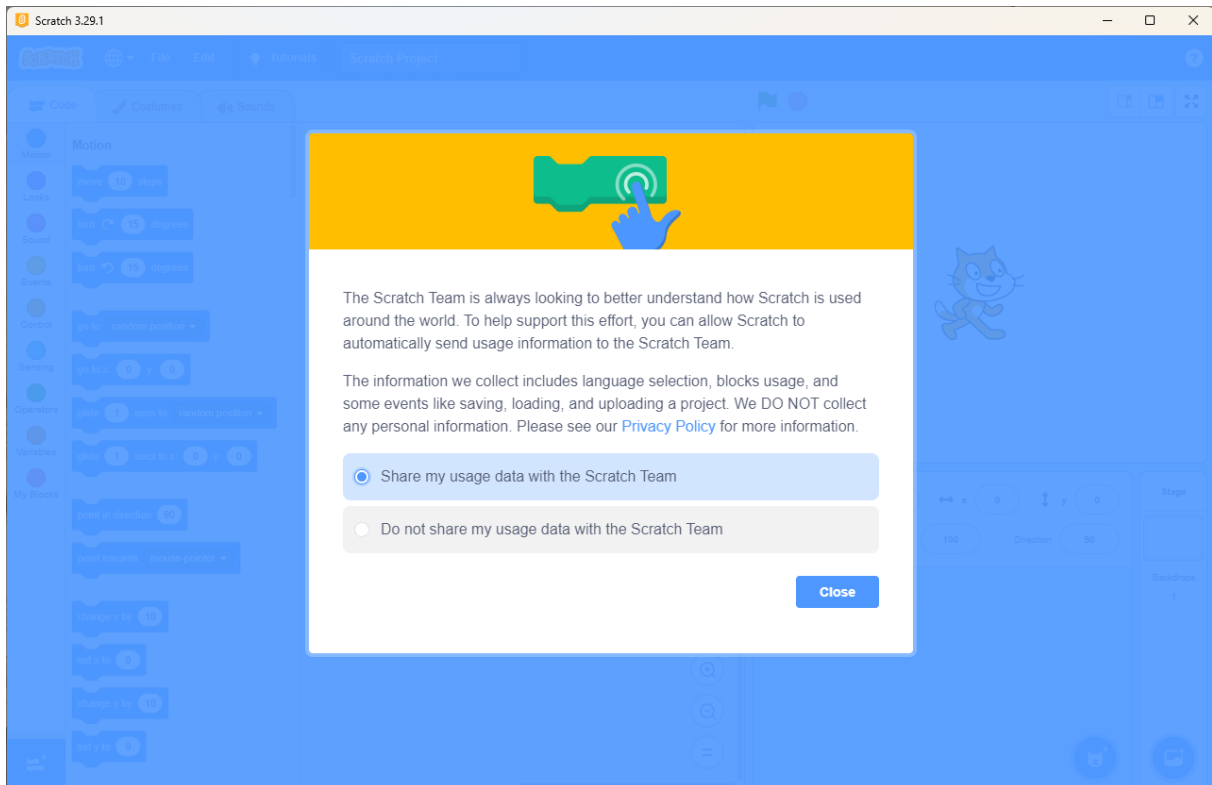


Installation window after installation completed.

Congratulations you installed Scratch. On your Desktop you will find an icon to open the scratch editor again. Do not delete this icon otherwise it will be very difficult to reopen the scratch editor. But don't be scared you can restart or shutdown your computer anytime or use any other programs.

1.1 Using scratch for the first time

After you clicked on Finish the scratch programming interface starts. If you are using scratch the first time it will ask you about your usage data. If you agree to share your data with the Scratch Team what includes language selection, block usage and some more impersonalised data select the first option. If not, select the second option. If you want to know more about the scratch privacy policy click the link provided. If you are not sure what to choose, take the second option.



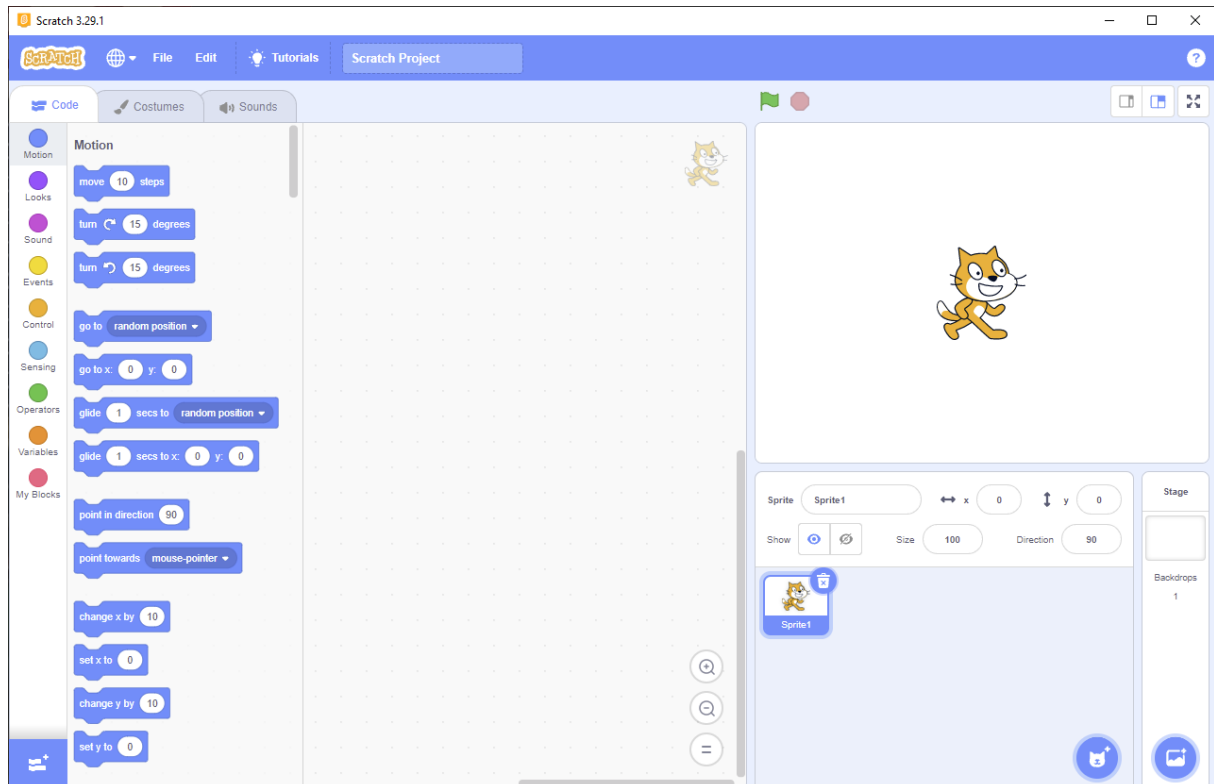
Scratch asking for permission to collect personalised data.

If you have problems installing Scratch contact the teacher. The contact details can be found in the previous section. If you have any questions about the successful installation process write them in the intended space and continue with the next chapter.

1.2 Question space

2 Introduction into Scratch

In this chapter you will learn the different program sections of the scratch programming Editor introducing some new concepts to describe the sections. Use the image below to localise the section or open your scratch editor. Let's explore the world of Scratch!



Overview of the scratch programming interface.

2.1 Header

The header of the editor is located at the top of the window and provides the project settings. You do not need to memories all these functionalities or where they are located but it is important to know that they exist. A programmer follows the principle of knowing where to find the knowledge.

The functionalities of the header are listed below as accessible from left to right.

Language Selection: After the scratch logo there is an icon looking like a world with longitudes and latitudes. Click this icon to select the language. Select English if not already ticked.

New, Save and Load: The item “Edit” provides operations to save and load your projects.

New: Replaces your current project with a new empty project.

Load from your computer: Opens dialog box to import any project stored. This option can be used to check your solution with the authors solution by importing the solution of the author.

Save to your computer: Opens the dialog box to specify the storage location of your project. Before you can store your project, you need to name your project. How to name your project is written below.

Edit configurations: The item “Edit” provides operations to restore changes and enables turbo mode. You barely use this functionality.

Restore: Restore the last deleted sprite or backdrop.

Turn on/off Turbo Mode: Toggles the turbo mode. We do not need the turbo mode. If you are interested in the turbo mode, you find a documentation online on the scratch wiki (https://en.scratch-wiki.info/wiki/Turbo_Mode).

Tutorials overview: The item “tutorial” provides a palette of simple tutorials to get used to scratch. Try some of them after you completed this script.

Name your project: The item “Scratch Project” is an input field where you can write the name of your project.

About the software: The question mark icon on the right provides help about scratch.
About: Clicking on the “About” option will open a window with version details of your scratch program. If the version is 3.29.1 you installed the correct version.
Privacy Policy: Clicking on the “Privacy Policy” option will open a window with the privacy policy of scratch.
Data Settings: Clicking on the “Data Settings” option will open the dialog box where you can change your data settings you made the first time you used scratch.

2.2 Block Palette

Let’s move on to the block palette section what is located on the left side of the programming interface. As mentioned earlier Scratch is a block-base programming language. This means that every command to run a program has his own block. Scratch has 6 different blocktypes what can be differentiated by its shape. To understand the block-coding technique it is important to know the differences between the blocktypes and where to use what kind of block.

2.2.1 Block Shapes

2.2.1.1 Hat Blocks

A hat block is always located at the top of a script. All commands located to a script must be under a hat block what can be compared to your body. It is not possible to add blocks on top of the hat block. Scratch knows 6 hat blocks what most of them we go to discover in the next chapters.



Hat-Block

2.2.1.2 Stack Blocks

The stack block is the most common block we use. There are over 60 stack blocks in scratch. As the name indicates, the stack block can be placed under or over a block what provides a bump or notch. The stack block does contain the magic of programming by performing a specific command.



Stack-Block

2.2.1.3 Boolean Blocks

The Boolean block is a condition block what represents ether true or false (Example: Is today your birthday? Ether true or false). The boolean block looks like an elongated hexagonal shape and can be inserted in holes with the same shape. A boolean block cannot be used as a stack block and must be inserted inside another block.



Boolean-Block

2.2.1.4 Reporter Blocks

The reporter block is representing a value. The reporter blocks are used for calculations (addition, subtraction, multiplication, and division) and variable values during runtime. The reporter block does have round corners and can be inserted in holes with the same shape.



Reporter-Block

2.2.1.5 C Blocks

C-Blocks are probably the most important blocks in scratch. C-Blocks are used for loops and Conditions. The required script for a loop or the condition is included in the “mouth” of the C-Block. Therefore, the block does look like a c the block was named after it.



C-Block

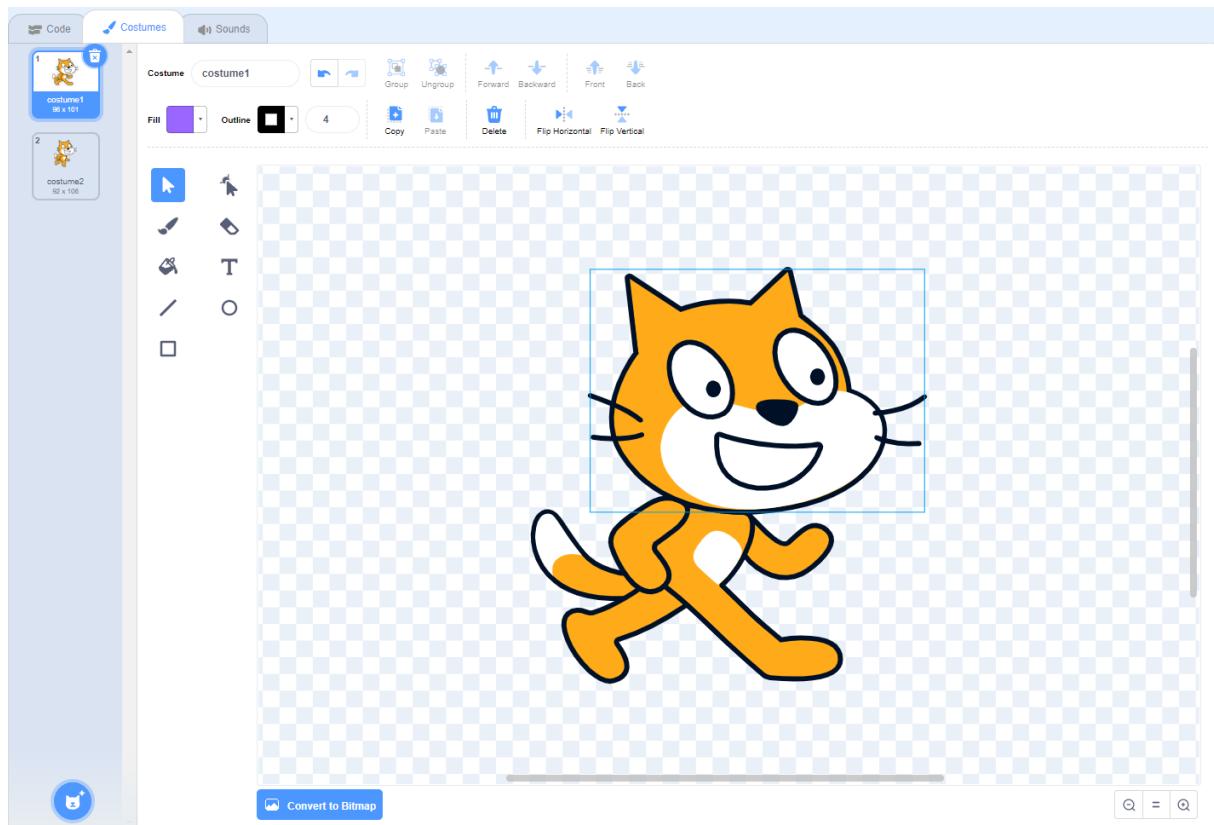
2.2.1.6 Cap Blocks

Cap-Blocks are barely used but they provide some special features we will have a look at in the chapter Events & Controls. Cap-Blocks can be found at the bottom of scripts where nothing can be appended. It is basically the opposite of the hat-block.



Cap-Block

2.3 Paint Editor



Scratch's built-in image editor

The paint editor makes scratch different from many other coding editors and provides basic image editing options. This editor is used for costumes, representing your sprites, and backdrops.

The editor supports vector graphics and bitmap graphics what can be converted into the opposite format. The difference between those two formats is simple. Imagine a red circle at the center of the image. For a vector graphic you store the shape, the position, and the colour of the figure. For the bitmap graphic you will store for each pixel (square on your screen) his own colour. If you fit all pixels in the right order in a grid you receive the image. Vector graphics are used for drawings whereas bitmap graphics are used for photographs. If you create a new sprite, costume or backdrop always consider what graphic format fits best. You cannot choose a false graphic format, just a more appropriate or inappropriate one.

In the left navigation panel of the editor, you can select the costume you want to edit. One or more costume will result in a sprite. If you want to add a costume hover the Cat-Face Icon at the bottom of the navigation panel. You can now select to upload an image as a costume or the draw a costume from scratch. If you want to use a predefined costume, click the magnifying glass. When you click the surprise icon, a random costume of the predefined costumes is selected. To delete a costume, select the costume and click the trashcan in the top right corner. If you are a bit confused about costumes, don't worry we will have a closer look at costumes and backdrops, what are also editable in the paint editor, in chapter 7.



Choose a Costume

There are a lot of images editing options what we won't have a closer look at. If you need to use one of these options, you can hover the icon what provides a tooltip or look them up in the scratch wiki.

This is just a small description of the paint editor. If you are fascinated by the topic, you will find advanced documentation on the scratch wiki (https://en.scratch-wiki.info/wiki/Paint_Editor). But do not spend too much time in reading the documentation, we learn by doing.

2.4 Sound Editor



Scratch's built-in sound editor

The Sound Editor can be opened by clicking on the tab next to the costume tab. The Editor provides options to edit and remix your sounds. The editor is split into a sound pane on the left, where you can select the audio to edit, and the Editing area in the center where you can remix and edit the audio.

Wherefore can a sound editor be used while developing a game? Nearly every game has a background music and some special sounds if you click a button. You can modify easily and fast your sounds in the sound editor to create a unique and recognizable gaming experience.

Like the paint editor you can import, load, or record you own audio by clicking the speaker button at the bottom of the sound pane. Scratch provides a bunch of sounds what most of the time are perfect. To delete a sound, select the sound and click in the right to corner the trashcan.

We will barely use the sound editor if you want to get familiar with it visit the scratch wiki (https://en.scratch-wiki.info/wiki/Sound_Editor) where all functionalities are described in detail.

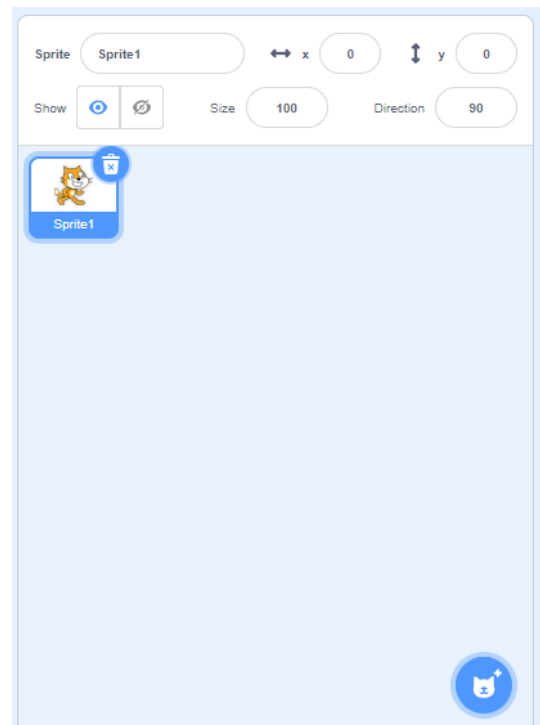
2.5 Sprite Pane

The sprite pane contains all sprites. Sprites are figures used for your game.

As you can see you can give your sprite a name. Choose an intelligent name what represents what the sprite is doing. For example, if you have a cat moving from the left to the right a good name would be “*Cat moving from left to right*”. In the sprite pane you can also define the default position of the sprite with x and y axis. We will have a close look at the coordinate system later. You can specify if the sprite should be visible on the stage or not and you can define the Size of the sprite as well as the direction it is pointing. You will find mor details about this in the next chapter.

To create a sprite, click the cat-face icon at the bottom right of the pane and select your preferred option like adding a costume. If you click the sprite and then move to the costumes tab where you can add, remove, or modify all costumes of your sprite.

Click the trashcan in the top right corner of the sprite representation to remove the sprite including all costumes and scripts.



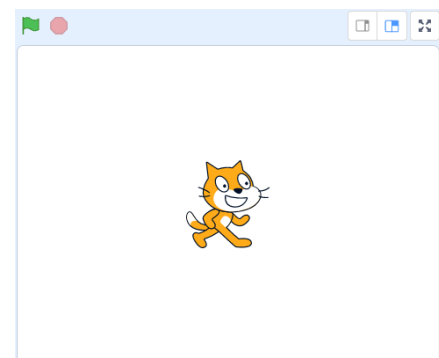
Sprite pane

2.6 Stage

The stage is located at the top right corner of the scratch interface and hosts the graphical result of your scripts. With the green flag you can start your program and the red button will stop it. In the upper right corner, you can choose what size the stage should be (small, large (default) or Fullscreen).

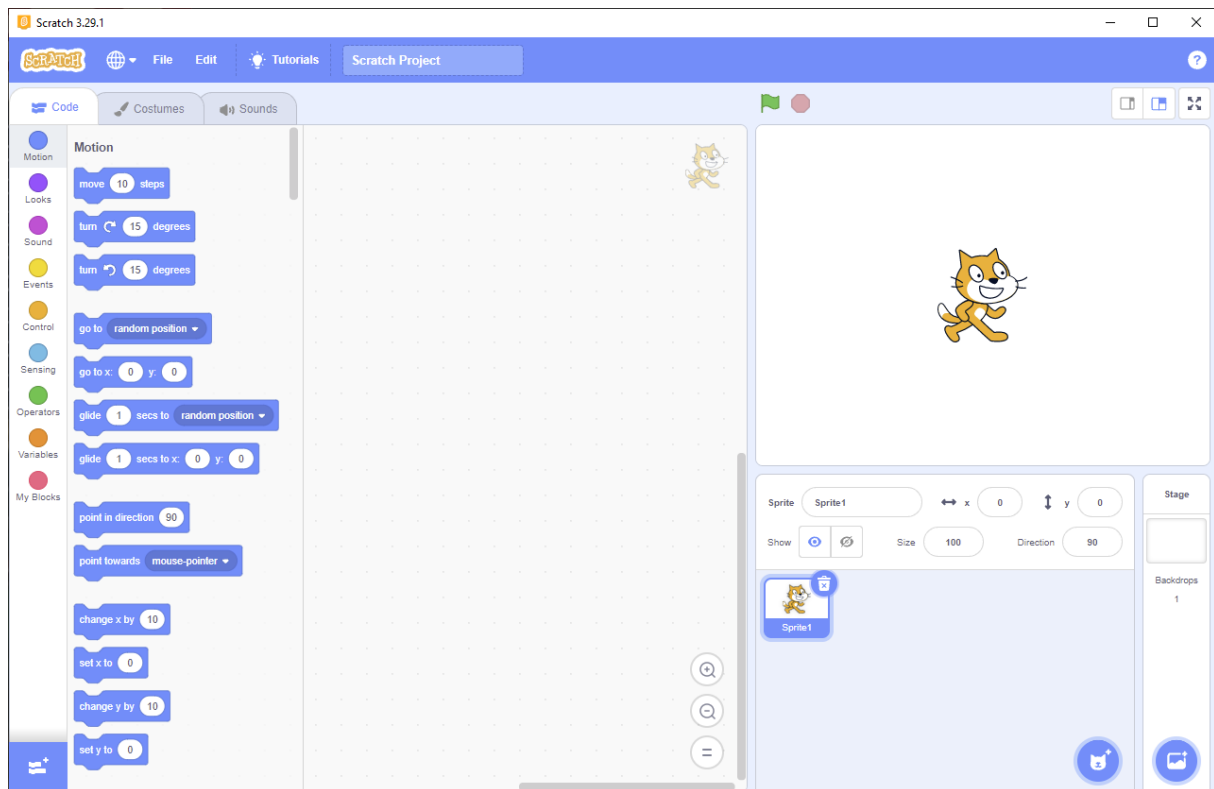
The stage is the back layer of all elements and has some special properties. A stage is stationary, what leads to the effect that the stage cannot move, change the size, or do any action related to its position. Sprites can talk to the user with bubbles, but stages cannot, asking questions to the user is no problem. The quintessence of this text is, that the stage provides less functionality than a sprite. In conclusion this means that not all blocks what can be used to write a script for a sprite can be used to create a stage script. There are a couple of more restrictions for a stage what must not be learned by heart and can be check on the scratch wiki if needed (<https://en.scratch-wiki.info/wiki/Stage>).

The stage can take on multiple backdrops. Backdrops are like sprites. They can be created, edited, and removed in the stage pane at the bottom right corner. A backdrop can contain scripts that are executed while the backdrop is active. But in comparison to the sprite a backdrop cannot contain costumes. We will have a closer look at backdrops and their usability in chapter 7.



Stage Pane

2.7 Code Area



Overview of the scratch programming interface.

The code area is the heart of the scratch programming interface where the magic of programming happens and can be found in the center. If you cannot see the code area make sure you are in the code tab and not using the paint or sound editor.

To create a script for a sprite or a backdrop drag your preferred block and drop it in the code area. To connect two blocks, move the second block as near to the first block as the grey preview block appears. The code area provides by a right-click a bunch of options what are very useful.

Right-click on code area:

Undo:	Reverts the last edit
Redo:	Replaces the last undo edit
Clean up:	Organizes all your scripts vertically
Add comment:	Adds a comment
Delete blocks:	Removes all blocks in the code area

Right-click on block:

Duplicate:	Duplicates the script what the block belongs to
Add comment:	Adds a comment
Delete Block:	Deletes the block including all contained blocks

A note on comments: Comments are the key to good code. I will explain this to you with a little story. Imagine you are starting a new project today but therefore it is very big you are not able to complete it in one day. After half a year you once again find time to continue with your amazing project. If you have no comments or useless comments you need to understand every script you wrote. With useful comments you can easily understand what purpose each script fulfils. What are good comments? A good comment does not describe what the code fulfils, it describes what purpose the code is used for. Let's use an example: You want to create a script moving the Sprite 20 pixels to the right. A good comment would be "Moving to start position". A bad comment would be "Moving Sprite 20 pixels to the

right” because all the comment says can be taken from the script which every programmer, what you are, easily understands.

2.8 Quiz

What editors does scratch provide?

- Paint Editor
- Sound Editor
- Text Editor
- Video Editor

What block type is used at the top of every script?

- Stack Blocks
- C Blocks
- Hat Blocks
- Cap Blocks

What is a sprite?

- A background image
- A configuration option
- A character
- A group of blocks

How can you start your game?

- Press the s key on your keyboard
- Press the green flag
- Double click the code area
- Press the red button

What is a script?

- An audio
- A C-Block
- A group of costumes
- A group of blocks

Solutions can be found in the package file 2_01.txt

2.9 Question space

3 Motion

Let's start and do some magic with scratch. This section is all about moving and turning around the famous cat sprite. The commands to move the sprite can be found in the motion category of the code section. All block categories are filled with a specific color what makes it easy to differentiate them. The motion blocks are filled with blue. Motion blocks are mostly stack blocks but there are three reporter blocks what represent position values of the sprite.



Scratch Cat

3.1 Move and turn around!

To move and tourn your sprite scratch provides a bunch of commands what can be found in the upper section of the block category motion. I recommend you to open the scratch programming interface and try what is explained. Therefore, grab the block and move it to the code area. Edit the values and double click the block to execute it.



Move () steps block

The simplest scratch block is the **move () steps** block. You can put a numeric value inside the brackets (in scratch put the numeric value inside the white gap). A numeric value can have many different expressions. Let's have a look at them.

Positive Number: This is the most obvious value. If you use any positive number, the sprite will move as specified steps in the direction it is facing. By default, this will be a movement from left to right.

Negative Number: If you specify a negative value with a leading - sign, the sprite will move in reverse direction as it is facing. By default, this will be a movement form right to left.

Floating Point Number: You can use floating point numbers as ether positive or negative value. The floating-point value is separated by a dot. If you check the value in the sprite plan you will see a false value while working with floating point numbers. But this is only a presentation issue. The real value is stored correctly in the background, where the sprite pane rounds the value what causes 0.5 to look like 1.

Scientific Number: If you try to add a text as steps the letters will not be accepted except for the letter e. But why is this? There is a simple answer, e is used in scientific notation and represents 10 to the power of x. The expression $0.5 * 10^2$ can be written as $0.5e2$ and would be allowed as a value.

As we have seen you can use numeric values, but you can't use any calculations inside the gap. Even dough the value 5-4 is accepted it will not run. We will find a way to use calculations inside the gap in section about operators.

A very nice behaviour of the **move () steps** block can be observed when it hits the border of the stage what prevents your sprite to move into the nowhere. Even if you use a very large number, it will stop at the border of the stage.

Now our sprite can move to the left and right but what if we want to move our sprite to the top or bottom of the stage or any other direction? There are two blocks in scratch what provide the same functionality in a slightly different way. Ether, you use



Turn right () degrees

the **turn right () degrees** block or the **turn left () degrees** block. You can use a numeric value inside the brackets (in scratch put the numeric value inside the white gap)

Those two blocks are almost equal. Let's explore this with an example. To turn your sprite 90 degrees to the right, you can use the **turn right (90) degrees** block, but you could also use the **turn left (-90) degrees** block or even **turn left (270) degrees**. The reason this is possible is because of the circle definition. A circle has 360 degrees what is equal to 360 pieces of the same size. If you now take the first 90 pieces of the circle to the right, you will be at the same position as if you take the first 270 pieces of the circle to the left or the first -90 pieces of the circle to the left. The

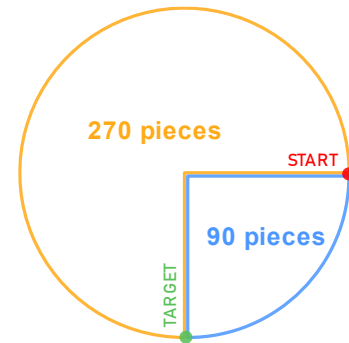


Illustration of left and right turn

image on the right side illustrates the the described redundance. Mathematically you can calculate *degrees to the right* = 360 - *degrees to the left* or *degrees to the left* = 360 - *degrees to the right*.

If you combine the move and rotate block you can move your sprite in any direction at any distance. But be aware, if you move your sprite 1 step 45 degrees to the left, the new coordinates will not be (1,1) because the sprite does not only move to the right, it also moves to the top. If you are interested in this special behaviour have a look at the law of sines on Wikipedia what will result in $y = \frac{\text{length} * \sin(\alpha)}{\sin(90^\circ)} = \text{length} * \sin(\alpha)$ and $x = \frac{\text{length} * \sin(180-\alpha)}{\sin(90^\circ)} = \text{length} * \sin(180 - \alpha)$.

3.2 Coordinates

We already know how to move a sprite but how can we ensure that the sprite starts at the right position? There are a bunch of function what position your sprite correctly on the stage.

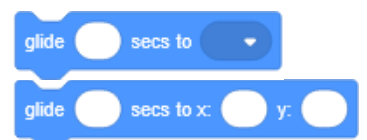
The stage, where the result is displayed, has a fix size of 480 pixels wide and 360 pixels heigh. The origin (0,0) of the coordinate system is at the center of the stage. In conclusion your sprite can take any position between (-240, -180) at the top left to (240,180) at the bottom right to be fully visible and provides a unique position.

Use the **go to x: () y: ()** block to position your sprite anywhere in the stage. As values you can use the same numeric values, if your value is too high or too low the sprite will stop at the border of the stage. If your sprite should glide to a position, there is a built-in block called **glide () secs to x: () y: ()**. Specify a moving time in the first white gap in seconds. You can use the same numeric values as in the **move () steps** block except a negative value. If you use a negative value, the **glide () secs to x: () y: ()** block behaves like the **go to x: () y: ()** block.



go to stack blocks

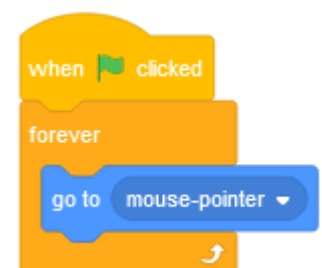
As you may have noticed there are the same blocks twice but with different white gaps, what are also called arguments. You have the possibility to position your sprite ether with gliding or directly to a random position or your mouse pointer. Therefore, use the block **go to (random position/ mouse pointer)** or **glide () sec to (random position/mouse pointer)**.



Glide to stack blocks

The block **go to (random position)** or **glide () sec to (random_position)** moves your sprite to a random position every time you execute the block. Can be used to generate enemies at random positions.

The **go to (mouse pointer)** block or **glide () sec to (mouse pointer)** will position your sprite at your mouse pointer. To demonstrate the **go to (mouse pointer)** block or **glide () sec to (mouse pointer)** use the following script on the side (Can also be found in the package at `/examples /3_mouse_pointer_follow.sb3`). Click the green flag at the top left of the stage and move your mouse on the stage. The sprite should follow your mouse what is a common use case in many games. We will have a close look at the other blocks in later chapters.



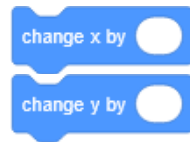
Mouse pointer script

But what is if you only need to modify either the y or x coordinate. There are the blocks **set x to ()** and **set y to ()**. You can use any numeric value, but they are limited by the size of the stage. The blocks are especially useful when you need to move your sprite up and down or forwards and backwards to a specific height or width.



Set to stack block

For the blocks **change x by ()** and **change y by ()** the current position is used in the background. These blocks are very similar to **move () steps** in combination with **rotate left () degrees** with a value of 0 or 90 degrees. You can use any numeric value as in the **move () steps** block, but if the border of the stage is reached it can only move in reverse direction.



Change by stack block

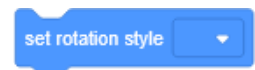
All these blocks will modify the position of the sprite with directly using the current position, but how can we get the current position of the sprite? You may have noticed the reporter blocks at the bottom of the motion block category. The reporter block **x position** and **y position** represent the current position. You can insert the reporter block in all white gaps what have the same shape as a reporter block. For example, you can use the **change x by ()** block and fill the white gap with the current x value by using the reporter block **x-position**. If you click the checkbox on the left of the reporter block, the current value is provided in the upper left corner of the stage what can be useful while checking if your script is doing what it should do and to see the floating-point numbers.



Position reporter block

3.3 Where to point

We know that it is possible to rotate a sprite. This feature comes with the problem, that if you rotate by 180 degrees the sprite is upside down. Scratch provides a solution to this problem. The **set rotation style ()** block allows to set a rotation style as following.



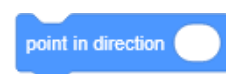
Set rotation style stack block

All around: This is the default value and will turn your sprite all around

Left-right: This value will flip your sprite (reflect) your sprite at the breakpoint of 0 and 180 degrees what will solve the upside-down problem. But when moving the sprite, it will move into the given direction. If your pointing at 200 the sprite will be reflected and while moving it will move to the bottom left corner of the stage.

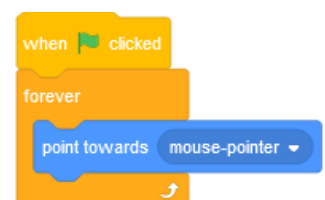
Don't rotate: This value will not rotate your sprite but when setting a point direction, the sprite will move in this direction. This option is like the left-right option but without any visible rotation.

To set the rotation use the **point in direction ()** block. You can use any numeric value in the white gap but remember a circle is divide into 360 pieces what means that 0 and 360 are the same value.



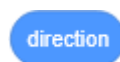
Point in direction () stack block

Some games do not only follow your cursor they do also point towards your mouse pointer. Therefore, the **point towards (mouse-pointer)** method can be used. Let's modify our previous script what was used to follow the mouse pointer to point always in direction of the mouse pointer. The script can be found on the right or in the package at `/examples /3_mouse_pointer_direction.sb3`. When clicking the green flag, the sprite should always point towards your mouse pointer, independent If you hover over the stage or not.



Mouse pointer script

To get the current direction at what the sprite is pointing the reporter block **direction** can be displayed in the stage with detailed value or even used in any white gap with the same shape as a reporter block.



Direction reporter block

3.4 Exercises

Exercise 3_01: Move 100 Steps to the right.

- Exercise 3_02:** Move 100 Steps to the left.
a) One command
b) Two different commands
- Exercise 3_03:** Rotate the sprite by 50 degrees to the right.
a) With the **turn right () degrees** block
b) With the **turn left () degrees** block
- Exercise 3_04:** Rotate the sprite by 90 degrees to the left and move 50 steps.
- Exercise 3_05:** Position your sprite at $x = 200, y = 100$
- Exercise 3_06:** Position your sprite at $x = 100, y = 50$ and set x to 20
- Exercise 3_07:** Position your sprite at $x = 100, y = 50$ and decrease x by 20
- Exercise 3_08:** Let your sprite glide 5 seconds from $x = -100, y = 50$ to $x = 100, y = -50$
- Exercise 3_09:** Set rotation style to right-left and point in direction 200 degrees.
- Exercise 3_10:** Set rotation style to don't rotate and point in direction 45 degrees and move 50 steps.

3.5 Question space

4 Operators

You already learned about moving your sprite. But what happens behind the scenes? Running a program is all about calculations with binary numbers. In this section we will have a look at calculations and comparisons what blocks can be found in the operators block category. As understanding decimal numbers is a lot easier, we will continue to use them.

4.1 Calculations

In school the first 4 operations you learn are addition $() + ()$, subtraction $() - ()$, multiplication $() * ()$, and division $() / ()$ what are represented in scratch by the first 4 blocks of the operators block category. The basic operation blocks are reporter blocks and add, subtract, multiply and divide two numeric values and report the result. You can use any numeric value inside the white gaps or any other reporter block.



Basic operations reporter block

Let's explore the usage of nested reporter blocks. You can connect multiple calculations by nesting the reporter blocks. But the calculation rules are not as you may expect. If you want to calculate $2*3+2$ you will first calculate $2*3$ and then add 2 what results in 8. But if you would use this in scratch you will use this nested block $(2) * ((3) + (2))$ what will result in 10. Nested reporter block gaps are executed from in to out. This means if you use first the multiplication block and nest the addition block. The addition block is executed first and after that the multiplication block is executed with the value reported from the addition block. To write this calculation in our common case you must use brackets. The calculation made by scratch would be written as $2*(3+2)$ what results in that the addition is done before the multiplication. If you need to do the calculation as displayed at the beginning, you must nest the multiplication block inside the addition block what results in $((2) * (3)) + (2)$.

As you know division of two numeric values will often result in a floating-point value. Maybe you want to use only decimal numbers. Scratch provides a simple block to fulfil your problem. The reporter block **round ()** will round up values that are .5 or higher and lower decimals are rounded down.



Round reporter block

Scratch also knows the modulo operator block. You may have never heard of this operator, but it is maybe the most useful operator in programming and certainly the most important operator for en- and decryption methods. The modulo operator block **() mod ()** is a reporter block. The modulo operation will return the remaining part of a calculation. Let's have a look at an example: The following script **(12) mod (5)** will return 2. This is because $2*5$ is 10 and there are 2 remaining. You may not see the benefit of this operator now, so let's see some examples when you can use the modulo operator block.



Modulo reporter block

Even or Odd: If you want to find out if a number is even or odd you can use **(x) mod (2)** what reports either 0 or 1. This can be useful if you want to draw a chess field or a table with row highlighting.

n-th time: If you want to do something only every n-th time you can use **(x) mod (n)** what reports values from 0 to n-1. If the value is 0 you are at the n-th time.

Time: If you want to display a time in a readable format. Imagine you have 140 Minutes and want to display the hours and minute you can use for hours **round ((140) / (60))** and for the minutes **(140) mod (60)** what results in 2 Hours and 20 Minutes.

Not sure if this is correct, try it out.

But there is an even more powerful mathematics reporter block in scratch. The **() of ()** block provides an enormous number of mathematic functions used to calculate positions of your sprite. The following functions are available, but don't worry you do not need to learn them by heart or to understand the mathematical concepts. They can be read online if required.

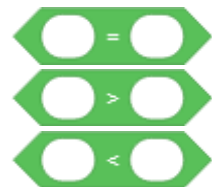
abs Reports absolute value
floor Reports the lower value

Example: -10 will report 10
 Example: 10.99 will report 10

ceiling	Reports the greater value	Example: 10.01 will report 11
sqrt	Reports the square root of the value	Example: 9 will report 3
sin	Reports the sinus of the value	Example: sin(90°) will report 1
cos	Reports the cosines of the value	Example: cos(0°) will report 1
tan	Reports the tangent of the value	Example: tan(0°) will report 0
asin	Reports the inverse sine of the value	Example: asin(1) will report 90°
acos	Reports the inverse cosines of the value	Example: acos(1) will report 0°
atan	Reports the inverse tangent of the value	Example: atan(0) will report 0°
ln	Reports the natural logarithm of the value	Example: ln(e) will report 1
log	Reports the logarithm of the value	Example: log(10) will report 1
e ^	Reports the e^x value	Example: e^1 will report e
10 ^	Reports the 10^x value	Example: 10^5 will report 100'000

4.2 Comparison

After calculating the value, you may need to compare it to another value what will lead to a different behaviour depending on the result. Scratch provides the comparison blocks in the operators block category as boolean blocks returning either true or false. There are the three basic mathematical comparison blocks equal (**() = ()**), greater than (**() > ()**) and less than (**() < ()**). You can use any alphanumeric value inside the white gaps or any other reporter block.



Equals, greater than and less than boolean blocks

If you use a numeric value, be careful with the greater than (**() > ()**) and less than (**() < ()**) block, because they do not include the value used in the white gap. What means **() < (100)** includes all values from -infinity to 99 and **() > (100)** includes all values from 101 to infinity. Always add (less than) or subtract (more than) one from the original value to include the value.

If you use an alphabetic value, be careful with case sensitivity. The comparison is case insensitive what means that a and A are equal. If you use greater than (**() > ()**) and less than (**() < ()**) blocks the alphabetical order is respected what concludes that a will be less than b because a is before b.

Scratch also provides blocks to connect boolean blocks what are also located in the operators block category. Let's explore these blocks with examples:

Is the sprite's x-position between 50 and 100?

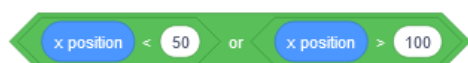
This problem looks very difficult but when you understand the concept it will be the easiest task in programming. We need to split the question into two comparisons. First, we check that the value is greater than 50 and second, we check that the value is less than 100. With the scratch (**() and ()**) block we can connect these two values what results in:



The (**() and ()**) block checks all values, if all boolean arguments are true the result is true, else it is false.

Is the sprite's x-position is less than 50 or greater than 100?

This problem can be solved nearly in the same way as the previous, but we need another keyword. First, we check that the x-position is less than 50 and second, independent of the previous result, we check that the x-position is greater than 100. With the scratch (**() or ()**) block we can connect these two values what results in:



The (**() or ()**) block checks all values, if one boolean argument is true the result is true, else it is false. It is the inverse of the (**() and ()**) block.

Is the sprite's x-position not less to 200?

This problem can be solved in various ways. Let's focus on the most obvious solution. Scratch provides the **not ()** boolean block. In comparison with the less than block we can solve this problem what results in:



The **not ()** block does negate every boolean response. This means a true is converted into false and reverse. But wait, isn't there a simpler solution. Yes, there is a much simpler solution provided by the **() > ()** greater than block what looks like **(x-position) > (199)**. This amazingly proves that there is not only one correct solution in programming. I recommend you choose the simplest solution.

These blocks are very powerful, but they can be even more powerful if you nest them. It is possible to connect the **() and ()**, **() or ()** and **not ()** block into itself. This brings the advantage that you can use the **() and ()** or **() or ()** block for hundreds of comparisons. Nesting the not block will not be that useful because this will inverse the boolean twice and end in the original value.



Nested and and or condition

4.3 String operations

Scratch also provides four basic functions to handle strings what are in the operators block category. Strings can be used to send messages to the user, and they are also the return type of the questions we ask the user, but we will have a closer look at user interaction in an upcoming chapter. Strings can contain any character (numbers, letters, special chars) except the new line character.

The **join () ()** reporter block does concatenate two strings together. Use two strings in the white gaps what should be connected. This block is useful if you ask the user for the first- and last name and want to display in the leaderboard the full name by concatenating these strings. As the **() and ()**, **() or ()** blocks you can nest the **join () ()** block to create long strings.



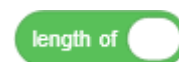
Join () () operator block

The other blocks provided by scratch do inspect an existing string. The block **letter () of ()** will return the letter at a certain position. Use for the first white gap a number what is between 1 and the length of the string. If you use any number outside this range an empty result is returned. The second white gap contains the string from what the letter must be extracted.



Letter () of () operator block

The block **length of ()** will count all characters passed in the white gap and return the count of the value.



Length of () operator block

The boolean block **() contains ()?** does check if the first white gap contains the substring of the second white gap. If the second white gap is empty, the block will always return true. The substring search is case insensitive. Therefore, the following blocks will return true: **(banana) contains (a)?** and **(banana) contains (A)?**



() contains ()? operator block

4.4 Exercises

- Exercise 4_01:** Calculate $(4+5)*(3-6)$ with scratch.
- Exercise 4_02:** Check if $3*5$ is greater than 14 and $5*6$ is less than 31.
- Exercise 4_03:** Check if the absolute value of -20 is greater than 0.
- Exercise 4_04:** Check if $15 \text{ mod } 2$ is equal to 1 or if $15 \text{ mod } 2$ is not greater than 0.
- Exercise 4_05:** Check if the string bracelet contains race and get the length of the string.

Exercise 4_06: Check if the third letter of adventure is v and if this letter is larger than w.

Exercise 4_07: Check if (a greater than c and if c greater b) or if (a greater than b and b is not less than c).

Exercise 4_08: Concatenate anti and dote and check if this string contains the substring ido and if the fourth letter is i.

4.5 Question space

5 Events & Controls

This chapter enables you to create complex and innovative tools. It prepares you to make decisions in your script what define when to start or what to do at a certain point depending on other values. As well you learn how to simplify and automate processes to make your life as a developer easier. The blocks used in this chapter can be found in the Events and Control block category where most of the hat blocks are located.

5.1 Events on actions

Previously we double-clicked each block when we wanted to execute it. With the hat blocks provided in the Events block category we can now stack one or more stack blocks and execute them at a predefined action. The most popular stack block is the **when flag clicked**. To use this block and any other hat block drag it to the code area and append one or more stack blocks or a single cap block. This block will be executed when the green flag in the stage is clicked. To stop the execution, click the stop button on the left of the green flag. This is used most often to start the program what often requires preparation such as set default values or change sprite costume and backdrops to start position.



When flag clicked hat block

Scratch also provides the **when () key pressed** block what executes the script after the requested key is clicked. You can choose from any of the following options:



When () key pressed hat block

Alphabetic letters:

The script will be executed if the requested character is pressed. There is no difference between upper- or lower-case characters because the upper-case character consists of the signal from shift and character a what contains the signal of the character a. The upper-case character is only a representation of the program you are using when shift and a is pressed.

0-9:

The script will be executed if the requested number is pressed. It is not possible to use any number larger than 9 because 10 consists of 1 and 0. A solution to this problem will be discussed in the chapter about sensing.

Arrow up/down/left/right:

The script will be executed if the requested arrow is pressed and mostly used to move the sprite in the direction where the arrow is pointing.

Space Bar:

The script will be executed when the space bare is pressed what is mostly used to pause and continue your program.

Any:

The script will be executed if any key is pressed on the keyboard and can be used to start your program (From start screen to interactive mode)

You can use any key for your game and this list is not exhaustive. For example often the letters w,a,s,d are used to replace the arrow keys therefore it is very important to document in a separate document (Word, Excel of whatever you prefer) what each key is used to and provide this information to the user when starting your program. This documentation is very important that you do not use the same key for multiple actions what can lead to a misbehaviour of your program.

Scratch also provides the hat block **when this sprite clicked** what executes its script when the sprite is clicked inside the stage. This block can be used to pause or continue a game, to select an option or many more.



When this sprite clicked hat block

The next hat block provided by scratch is executed **when the backdrop switches to ()** the backdrop changed to the selected. This can be used to reposition your sprite to a certain backdrop. I advise you to use this block rarely and only if there is no other option because you may use the same backdrop for multiple scenes what have different purpose. The script can now not identify if you are now in scene 1 or 2 with this backdrop.



When backdrop swithest o () hat block

Finally scratch provides the **when () > ()** block. This block is related to sensing blocks loudness and timer what will be discussed in the sensing chapter. Independent of knowing what loudness and timer is, this block executes the script if the condition **(loudness/timer) > ()** is true.



When () > () hat block

5.2 Handling Messages

Sometimes it is important that multiple sprites can communicate where scratch provides the broadcast solution. These blocks allow you to send and receive messages following the broadcast principle (When sending, you can not specify a receiver, the message will be sent to each sprite inside your scratch project).



Broadcast() stack block

To send a message use the **broadcast ()** block or the **broadcast () and wait** block. Both blocks send the same message, but the first block will continue its script while the second block waits until each script that was notified is executed and then continues its own script. When using the **broadcast () and wait** block you must be careful to not use the infinite loop, what is explained later in this chapter.



Broadcast () and wait stack block

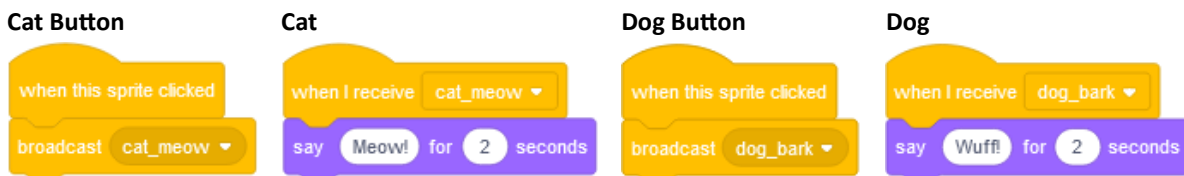
To trigger an action when receiving a message use the **when I receive ()** block. As on any other hat block append the required stack blocks to create the script that must be executed when the message is received.



When I receive () hat block

To create a message, select the option New Message in the dropdown of any of the broadcast blocks. Enter any name but follow a naming convention for your scratch project to prevent having multiple messages for the same purpose. Your previously defined naming convention should be documented in a document. You can use the same document for all your documentations.

This was a lot of theory, let's have a look at an example. Maybe you have a cat and a dog in your stage and two buttons. The first button requests the dog to bark, and the second button requests the cat to meow. With the naming convention `<to>_<action>` the script for the cat, dog and their buttons would look like this:



The whole script can be found in the package as example 5_message_example.sb3. Try to open the script and click the buttons and check if the messaging is working.

5.3 Loops

Now we know how to start a script, but often scripts contain repetitive code. Programmers never want to duplicate code because duplicated code requires to change multiple values what is hard to maintain. Scratch provides three c-blocks what can contain other stack blocks to avoid code duplication:

Repeat (): This block will repeat the passed stack blocks as many times as specified. After the execution of this loop, the script continues.

Forever: This block will repeat the passed stack blocks forever therefore you cannot append any stack blocks to this block

Repeat until (): This block will repeat the passed stack blocks until the condition returns true. This condition is created by any combination of operator blocks. After this succeeded the appended blocks will be executed.

What loop to use depends on the problem you must solve. The **forever** loop is useful for infinite actions such as moving the sprite to the mouse position or playing the same background music. The **repeat ()** and **repeat until ()** loops are used when you need a restriction. Depending if the restriction is on repetitions, use the **repeat ()** block, or on another boolean block, use the **repeat until ()** block.

Let's have a look at an example of a loop. Imagine your sprite must move on a circle. You can now add a block what rotates by 1 degree and moves 2 steps for 360 consecutive times, or you can use any loop block provided by scratch to solve this problem elegant and include this step only once. Imagine you want to make the circle bigger or smaller you need to modify the value of 360 blocks! The solution to this problem can be found as an example in the package 5_repeat_circle_move.sb3.



Repeat () C-block



Repeat until () C-block



Forever C-Block

5.4 Conditionals

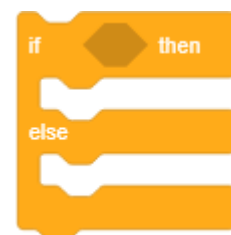
You already explored all Operators and many important boolean blocks but we never discussed where we can use these boolean blocks. The most powerful block in scratch are the **if () then** and **if () then else** blocks. These blocks decide, depending on your operators block combination, what scripts are executed and what not. Both blocks check if the given condition is true and execute the script if so. The second block does execute a different script if the condition was false.



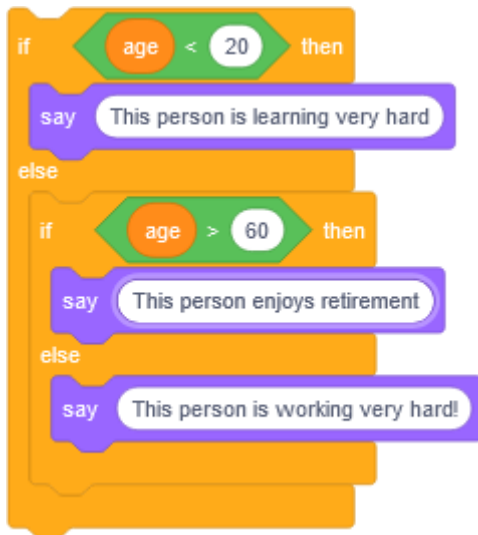
If () then C-Block

Imagine you ask someone how old he is. If this person is older than 60 you want to say "This person is very wise", what would solve the **if () then** block. But if he is not older than 60 and you want to say "This person is full of energy" you can use the **if () then else** block or use the **if () then** block twice with different conditions.

If you need a more detailed response. For example, for people younger than 20 you want to say "This person is learning very hard", for people between 21 and 60 you want to say "This person in working very hard" and for people older than 60 you want to say "This person enjoys retirement" requires to nest the conditions what results in



If () then else C-Block



5.5 Wait & stop

Remember the circle that the sprite was walking on? The sprite was walking at a predefined speed of the loop. To expand this time scratch provides the **wait () seconds** block. This block will wait with execution of the current script until the time expired. But this block can also be used for other actions such as showing instructions and start the game after 5 seconds etc.



Wait () seconds stack block

Imagine you have two sprites walking on the circle and the first sprite starts 3 seconds earlier, timed with the **wait () seconds** block. After walking half the circle the first sprite stops and continues walking when the second sprite reaches it. Therefore, scratch provides the **wait until ()** block what accepts a boolean block. Do not worry how to implement this, we just want to talk about the concepts and not how to identify if the second sprite reached half of the circle.



Wait until () stack block

Imagine your sprite is again walking on the circle infinitely while juggling. The juggling and the walking are separate scripts. In the upper right corner, there are three emergency buttons. The first button will stop juggling and walking at the same time, this can be realised with the **stop ()** block and the **all** option. The second button will only stop juggling what can be realised with the **stop ()** block and the **other Scripts** option. The third button will only stop walking what can be realised with the **stop ()** block and the **this script** option. An example nearly like the described problem can be found as an example in the package 5_stop_scripts.sb3.



Stop () cap block

5.6 Exercises

- Exercises 5_01:** Let the sprite move 100 steps and then turn 90 degrees to the right. Repeat this 7 times. What direction does the sprite face to?
- Exercise 5_2:** Let the sprite turn 45 degrees to the left and move 20 steps when arrow right is pressed. When arrow left is pressed do reverse (turn 45deg to the right and move 20 steps)
- Exercise 5_3:** Select a random number between 884 and 920 when the space bar is pressed. When the number is bigger than $21 \cdot 43$ move 20 steps to the right else 20 steps to the left.
- Exercise 5_4:** Create a button. If the button is pressed turn the main sprite 15 degrees to the left and move 10 steps.

Have this been difficult exercises? Don't worry, these exercises ask for previously learned knowledge what uses time to consolidate.

5.7 Advanced Exercises

The following exercises are very difficult and provided a step-by-step solution to complete the task. The exercises are introduced with the main goal followed by the detailed step-by-step solution while providing an insight to real life examples.

Exercise 5_05: Control the sprite.

Main Goal: Control the movement of the sprite with the arrow keys. Each key down should result in moving the sprite by 10 steps in the desired direction.

- Steps:*
1. Choose the **when () key pressed** hat block
 2. set rotation style (Up and down = all around; Left and right = left-right)
 3. Point in direction (90 degrees = right)
 4. Move 10 steps
 5. Repeat this steps for each arrow key and modify the parameter values

Exercise 5_06: Control the sprite with buttons.

Main Goal: Control the movement of the sprite with button representing the arrow keys. When pressing the button move the sprite by 10 steps in the desired direction.

- Steps:*
1. Create a new button (Tipp: Paint the button in scratch)
 2. Configure the button
 - 2.1 Choose the **when this sprite clicked** block
 - 2.2 Add the broadcast message block
 - 2.3 Create a new message for the direction
 3. Configure the sprite
 - 3.1 Choose the **when I receive ()** block
 - 3.2 Set rotation style (Up and down = all around; Left and right = left-right)
 - 3.3 Point in direction (90 degrees = right)
 - 3.4 move 10 steps
 4. Repeat this steps for each direction and modify the parameter values

Exercise 5_07: Don't touch the border.

Main Goal: Let the sprite move infinitely 10 steps. If the sprite touches the border change the direction by 55 degrees and continue, the sprite should always point left or right.

- Steps:*
1. Choose the **when flag clicked** hat block
 2. Set the rotation style to left-right
 3. Add the forever loop
 4. Add an if statement to the loop
 5. Create the border touch conditional what is added to the if statement

$$([\text{abs of } (x\text{-position})] > 190) \text{ or } ([\text{abs of } (y\text{-position})] > 110)$$

6. Turn 55 degrees inside the if statement
7. Stack to the if statement to move 10 steps

Solutions for this exercises can be found as usual in the package.

5.8 Question Space

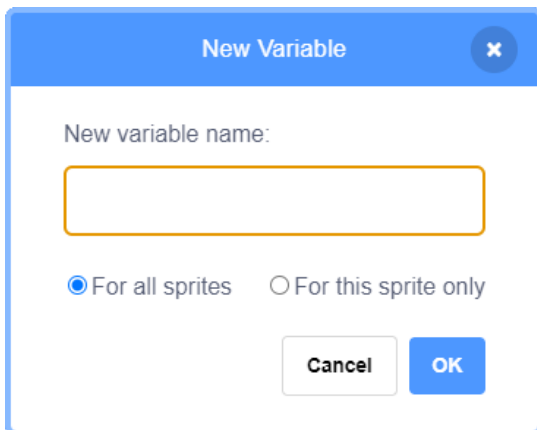
6 Variables & Lists

This section is all about variables and lists what provides the ability to store data in scratch's memory. This can be useful in many different situations such as storing the username, allocate points gained by a user and many more. The blocks used in this chapter can be found in the Variables block category.

6.1 Manage a variable

Variables are very useful when you want to store a single value. The value can be either a number or a string and is represented as a reporter block and often represent values related to a sprite.

Unlike all other blocks you met, you first must create an instance of this block by clicking "Make a Variable" what opens a new window to specify the variable details:



Popup while creating new variable

Variable Name: Each variable is identified by its unique name. There are never two variables with the same name. You can use any character you want but choose a self-descriptive and generic variable name. The name should be that clear that someone not related to your project will be able to get the purpose of the variable only reading the variable name. There is no problem to have a variable name containing multiple words.

Global Variable: Global variables can be set by choosing the "For all sprites" option. This means that you can access and modify the value of the variable from all sprites. This can be useful for game states or usernames what should be accessible by every sprite. But follow the principle of need-to-know access. If no one else needs access to the variable uses the local variable.

Local Variable: Local variables are also called private variables because they are only available to the sprite that the variable is related to. To create a local variable use the "For this sprite only" option. Local variables are commonly used for calculations or sprite states.

You can also create variables for backdrops what are accessible by all sprites. It is recommended to declare non sprite related variables in a backdrop. Before creating a variable always consider where to create and what visibility the variable must have. After filling in the form you can click ok to create your variable.

A variable can be controlled by four stack blocks. The block **show variable ()** displays the value of the variable in the upper left corner of the stage. This can also be achieved by clicking the checkbox on the left of the variable reporter block. But clicking the checkbox will display the value always if you have a highscore variable you may want to display the highscore only after the game is over, so you need the **hide variable ()** block to hide the variable and when the game is over you can use the **show variable ()** block to display the highscore.



Show variable () stack block



Hide variable () stack block

The advantage of those two blocks is that you can select the variable what is displayed. This makes it easy to display either the personal highscore or the overall highscore by switching the variable selection inside the **show variable ()** and **hide variable ()** stack blocks.

To set a value for your variable you must use the **set () to ()** block. This block overwrites the value of a variable to the number or string specified. If you use a number, you can use the **change () by ()** block to increment the value or decrement it by using a negative number, what is very helpful for lives or points gained in a game. If you have a string as a value, the **change () by ()** block will set the value of the variable to the value specified as increment.



Set () to () stack block



Change () by () stack block

To use the variable value, you can use the provided reporter block like the **x-position** or **y-position** block in the Motion block category. Often this reporter block is used inside calculation or comparison blocks depending on the value and purpose of the variable.



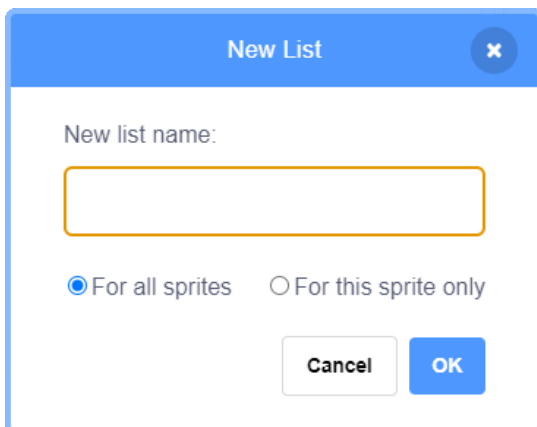
Reporter block

To rename the variable use any block managing the variables and select the option rename what opens again the popup window where you can define the new name. Set a self-descriptive name as explained above. If the variable is no longer needed, you can select the option “Delete the ___ variable” to delete the variable permanently. These options can be used while developing, during the game these options are not available inside any blocks managing variables blocks.

6.2 Manage a List

Variables are powerful and open a lot of innovative and challenging possibilities, but variables are limited when it comes to manage a lot of data. This is where the list will help you. Imagine you want to create a scoreboard. When using variables, you must create for each game iteration a variable what counts the points gained by the player. When using a list, you can write the score of each game into a separate space in the list and then ask for the score of a game in the list. The list will act as a container for the same datatype.

To use a list, you must create an instance of the list block by clicking “Make a list” what opens a new window to specify the lists details, like creating a variable.



Popup while creating new list

Set the list name using the same recommendations as for the variable name. But be aware that lists and variables are not the same therefore there might be a variable and a list having the same name what is not a problem. Select whether the list is globally or locally available by selecting either “For all sprites” or “For this sprite only” option. As for the list name follow the recommendations as from the variable options above.

As for variables you can also create lists for backdrops what are accessible by all sprites. It is recommended to declare non sprite related lists in a backdrop. Before creating a list always consider where to create and what visibility the list must have. After filling in the form you can click ok to create your list.

As lists are more powerful than variables, the list provides a bunch of additional blocks to administrate it. As for variables you can decide if the list should be visible on the stage or not. Use the stack block **show list ()** to display the list on the stage and **hide list ()** to hide the list from the stage. When developing it is recommended to display lists and variables for debugging by clicking the checkbox on the left of the variable reporter block. While your game is completed make sure that you hide all variables what should not be visible.



Show list () reporter block



Hide list () stack block

A list is a container for key-value based data where the keys are integers starting from one upwards. The integer keys are called index and are used to reference a value. This sounds confusing to you. Let's explore this concept based on an example. Consider a list containing names of employees. Each employee referenced by the index stores the employee's name. You might now want to add or remove an employee or even change the employee's name. All this is possible with lists.

To append a value to a list, use the stack block **add () to ()** where the first argument contains the value and the second argument the list where the value should be appended. If you want to insert a value at a certain index, you must use the **insert () at () of ()** stack block where the first argument is the value, the second argument is the index and the last argument is the list. You must use an index higher than zero and lower or equal to one more as the number of items in the list.



add () to () stack block



insert () at () of () stack block

To get the number of items inside the list use the reporter block **length of ()**. But be careful when using the **insert () at () of ()** block because inserting an item results in incrementing all indexes above the inserted index as the following example illustrates. Imagine a list with 10 items and a new item that will be inserted at index 5. This results in the behaviour that the item at index 5 will get the new index 6 and the item at index 6 will get the index 7 and so on until the item at index 10 will get index 11 what might results in accessing wrong values. More often you might use the **replace item () of () with ()** stack block where the first argument is the index, the second argument the list and the last argument the new value. This does not insert a new value but changes a value at the specified index. Use for the index a value greater than zero and less or equal to the number of items inside the list. If the given index does not reference a value nothing happens.



length of () reporter block



replace item () of () with () stack block

After inserting values, you may also want to delete them. To delete an item from the list you must use the **delete () of ()** stack block where the first argument is the index and the second argument the list. If the given index does not reference a value, no value is removed. If you want to clear the list and remove all items from the list, use the **delete all of ()** stack block. This block will clear the list but the list is still available and can be filled with new values again.



delete () of () stack block



Delete all () stack block

After we learned about how to store and delete values to and from a list, we focus on how to retrieve the data. As the data is stored in a key value pair the key is required to access the value with the reporter block **item () of ()** where the first argument is the index and the second argument the list. If you use a index that does not reference a value an empty result is returned. Scratch also provides the opposite operation. With the **item # of () in ()** returns the index of the value given in the first argument. The second argument defines in what list the value should be stored. If the value is not found zero is returned. But to avoid this error you can use the boolean block **() contains ()** where the first argument is the list and the second the searched value. Using an if statement with **() contains ()** and the body of **item # of () in ()** the zero will never be returned.



item () of () reporter block



item # of () in () reporter block



() contains ()? boolean block

To rename the list use any block managing the lists and select the option rename what opens again the popup window where you can define the new name. Set a self-descriptive name as explained above. If the list is no longer needed, you can select the option "Delete the ___ list" to delete the list permanently. These options can be used while developing, during the game these options are not available inside any blocks managing lists blocks.

6.3 Exercises

- Exercise 6_01:** Create a variable named “name” and assign your name as the value when the game starts.
- Exercise 6_02:** Create a list named “family” and add all family member names when the game starts.
- Exercise 6_03:** Create a list named “ShoppingList” and add Bread, Milk, Eggs, Apples, Chicken breast. After initializing the list replace the item 4 with Butter, then replace Milk with Rice and add Cheese at index 2. Finally remove the Eggs.
- Exercise 6_04:** Control the movement of the sprite by asdw (left, down, right, up) and log everything into the list log.

6.4 Advanced Exercises

The following exercises are very difficult and provided a step-by-step solution to complete the task. The exercises are introduced with the main goal followed by the detailed step-by-step solution while providing an insight to real life examples.

- Exercise 6_05:** Autofill a list
Main Goal: Create a list containing all numbers from 1 to 1000 in a simple way without repeating the code
Steps:
1. Create a list called “numbers”
 2. Add the **repeat ()** c-block and set 1000 incrementations
 3. Use the **add () to (numbers)** block
 - 3.1 Use the list length + 1 as the new value
- Exercise 6_06:** Replay the log
Main Goal: Replay the log previously created in Exercise 6_04
Steps:
1. Create variable named “logPointer”
 2. When flag clicked initialize replay
 - 2.1 Move sprite to center (0,0)
 - 2.2 Point in direction of 90°
 - 2.3 Set logPointer to 0
 3. Create repeat with **length of (logPointer)**
 - 3.1 Create if construct checking if log at logPointer moved up, right, down or left and point in desired direction
 - 3.2 Move 10 steps
 - 3.3 Wait 1 second to follow actions in replay use
 - 3.4 Increment logPointer
 4. Reset log
 5. Move to center (0,0)

6.5 Question Space

7 Looks

You have learned all about controlling the sprite inside the stage. This chapter will enable you to talk to the player, use color effects on your sprite, rearrange your sprite on the stage or switch costume and backdrops. All required blocks can be found in the Looks block category.

7.1 Talk to the player

We have learned a lot about controlling the sprite, but we never learned how to inform the user what he must do to control it. Scratch provides a bunch of options in the Looks block category. Let's start with the simplest block provided **say ()**. This block opens a speech bubble in the right upper corner of the sprite including the given argument as the message. The argument of the **say ()** method allows you to use any kind of symbols (letters, numbers, emoji's etc) and reporter blocks, such as variables or results of calculations. While the message is presented to the user the script immediately moves to the next block and continues. The speech bubble will only be removed if another speech bubble or thought bubble overwrites the existing one or by using the same block with an empty argument. Aswell if you stop the program by clicking the stop point the speech bubble will disappear. If you just want to show a message to a user for a specified time, use the advanced stack block, **say () for () seconds**. This block will show your message until the specified time is over. During this time, unlike the **say ()** block, your script is paused and executed after the speech bubble is removed.



Say () stack block



Say () for () seconds stack block



Think () stack block

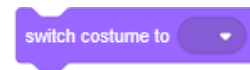
Maybe your sprite is not talking about something but thinking, you can use the stack block **think ()** and **think () for () seconds** what work the same way as **say ()** and **say () for () seconds** except instead of a speech bubble a thought bubble is used. These blocks can be very useful if you provide hints or recommendations to the user that should not be shown as talking messages.



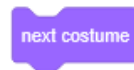
Think () for () seconds stack block

7.2 Change your look

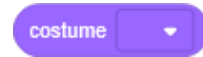
Let's talk about costumes what can be found in the Cosumes tab. Each sprite does consist of at least one costume. If your sprite does have more than one costume you can change the visible costume with coding blocks. This can be useful if you want to create an animation like talking, jumping, walking and many more. To introduce the concept of costumes we will have a look at an example by using the scratch cat moving to the right. It is not very difficult to code the movement to the right but naturally a cat does move the legs while walking. When selecting the scratch cat and then clicking on the Costumes tab you will see that the sprite provides two costumes. Scratch also provides an editor to modify each costume, but for our example we do not need to modify something. The block category looks provides the stack block **switch costume to ()**. For the argument of this block use any of the provided options or a reporter block. The value of the reporter block can ether be the number of the costume or its name. With this block we can implement the leg movement by following this algorithm: First we set the first costume to be visible by using the **switch costume to (costume 1)** stack block. When the correct costume is selected the cat should move 10 steps to the right by using the **move (10) steps** stack block. To see the different costumes, we need to implement a little brake by using the **wait (0.05) seconds** stack block. Now me need to change the costume by using the **switch costume to (costume 2)** stack block. After that move again to the right by using the **move (10) steps** stack block and wait again with the **wait (0.05) seconds** stack block. When putting this block of code into a loop what will repeat this actions 10 times you will have a smooth and natural movement of a cat.



Switch costume to () stack block



Next costume stack block

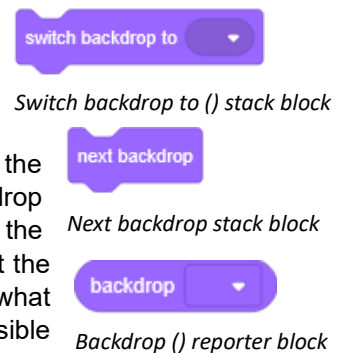


Costume () reporter block

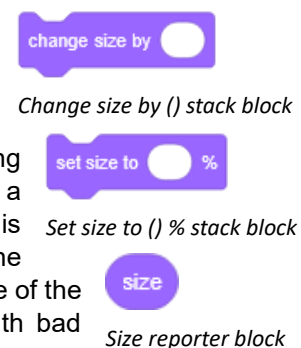
Maybe you realised that this could be even simpler, because the algorithm always selects the next costume, if there is no next costume it will go back to the first costume. To implement this even simpler algorithm scratch provides the stack block **next costume**. Both algorithms implemented in scratch are available in the example file `7_change_costumes.sb3` in the package.

If you need to retrieve the name of the costume or the costume number, you can use the reporter block **costume ()** inside the looks block category. This can be helpful when the sprite does have more than one costume, and you want to know at what point in the movement the animation is.

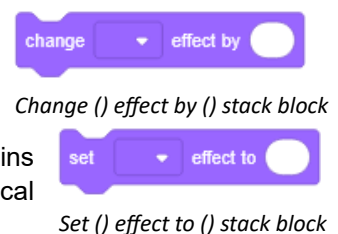
Independent if you want to create a game, story or animation you might want to have different backgrounds. Scratch also helps here with this. Click the stage pane on the bottom right. Then navigate to the backdrops by selecting the backdrops tab where all your backdrops are listed. You can modify, add, remove or draw the backdrops. In general, you have the same possibilities as for costumes. As costumes and backdrops are very similar the blocks are also quite similar. If you want to change to a custom backdrop scratch provides the **switch backdrop to ()** stack block what is working in the same way as **switch costume to ()** stack block. If you want to implement the second algorithm explained before use the **next backdrop** stack block what works the same way as **next costume** stack block. To get the current visible backdrop use the reporter block **backdrop ()** where you can select between backdrop number or name. This block is the equivalent to **costume ()**.



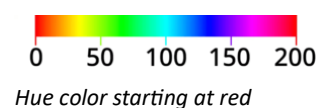
You can now change the sprites costume and the underlying background. If you once need to change the size of the sprite, as for growing or shrinking object, scratch provides you two stack blocks to accomplish this. To change the size of the sprite relatively to the actual sprite size use the **change size by ()** stack block. As an argument use any number or a reporter block reporting a number. If you use a positive number your sprite will increase, if you use a negative number it will shrink as the value is added to the current size what is available in the **size** reporter block or in the sprite pane. If you want to set the sprite size directly use the **set size to () %** stack block where the default value of the argument is 100. Be carefully with those blocks, if you have an image with bad resolution your sprit might looks pixelated.



But that is not all scratch provides to change the look of your sprite. With the stack blocks **change () effect by ()** and **set () effect to ()** you can customize your sprite with different graphical effects. The first argument requires one of the seven different graphical effects either selected from the dropdown list or from a reporter block. The second argument contains number specifying the amount of the graphical effect. The following graphical effects are available:



Color: This effect changes the color of the sprite where each costume can take on 200 different colours. The initial color of the sprite will have the value 0 and is reached again at 200. When your sprite is red then at value 100 your sprite will be blue, but when your sprite is another color, the value 100 will result in a different color. When using this filter for white and black the colours will remain and do not change as they are no colours.



Fisheye: This effect gives the impression of a sprite being seen through a wide-angle lense. Use any number between -100 and $3.4 * 10^{24}$ to have an effect. If you use a value outside this range the maximum effect is used.

Whirl: This effect twists the sprite around its center point, therefore distorting the sprite. This effect will behave differently depending on the operating system and browser. It is not recommended to use this effect. If the value used gets big enough the effect will generate an oval confined by the bounding box of the sprite.

Pixelate: This effect pixelates the sprite. This effect does not know any limits but at some point, a value incrementation will not lead into a different effect as there is no fewer than one pixel. If you use a negative number, the absolute value is used.

Mosaic: This effect shows multiple smaller images of the sprite, therefore creating a mosaic effect. The maximum value that can be used is 5105 what leads to a maximum of 262'144 duplications. If you use a negative number, the absolute value is used.

Brightness: This effect changes how desaturated the sprite is. Use a value between -100 and 100. At -100 the costume will be entirely black and at 100 the costume will be entirely white.

Ghost: This effect modifies the transparency of the sprite. Use a value between 0 and 100 where the value 100 hides the sprite but it is still detectable in some ways what can be useful for hidden platforms or chests.

You can use multiple graphical effects on the same sprite or costume. To reset all graphical effects scratch provides the **clear graphic effects** stack block. Still not sure what the effects will do? Nothing goes over trying



Clear graphic effects stack block

7.3 Move to spotlight

The filter ghost enables you to hide and show a sprite, even make the sprite permeable, but the sprite is still detectable in some ways we learn later in this chapter. Scratch provides the two stack blocks **show** and **hide** to show and hide sprites. To verify if the sprite is hidden or not you cannot only check the stage for visibility of the sprite, as the filter ghost could hide it, you also need to check the icons below the spite name in the sprite pane. If the left icon is selected (blue) the sprite is detectable on the stage else, it is hidden.



Show/hide stack block

Scratch also provides stack blocks to arrange the sprites on the stage. By default, the latest added sprite is in front of all other sprites. To move a sprite to the very frontmost or to the backmost use the **got to () layer** stack block. Use ether front or back as the value of the gap from the given dropdown list. If you have multiple sprites and want to move only a specified number of layers in front or to the back use the **go () () layers** block, where the first argument is selected from the dropdown specifying the direction of ether forwards or backwards. The second argument is a positive number what specifies the number of layers the sprite should move. When using the two blocks after each other it is possible to move a sprite to a specific layer independent of its current layer. Therefore, you must specify where to start counting layers, ether from the back or from the front. For example, if you have five layers counting from the back and your sprite must go to layer three use the **go to (back) layer** followed by **go (forward) (3) layers** stack blocks.



Go to () layer stack block



Go () () layers stack block

7.4 Exercises

Exercise 7_01: Say hello to the user when the game starts.

Exercise 7_02: Create two buttons and hide them when they are clicked

Exercise 7_03: Add the walking bear sprite from the scratch library. Create a walking animation where between each costume are 10 steps and 0.05 seconds of waiting time. In total walk 240 steps.

Exercise 7_04: Add the bat sprite from the scratch library. Create an infinite flying animation with a waiting time of 0.1 seconds.

Exercise 7_05: Add the dinoaur2 sprite from the scratch library and change the color to blue by using the right hue value (Do not try, see script section) and set mosaic mode to have 4 dinosaurs.

7.5 Question space

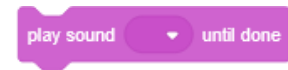
8 Sound

You learned how to use graphical effects on your sprite what makes your project more colourful. Humans have 5 senses so we completed the seeing part. Let's move on to the hearing part and add some sounds to your amazing projects. All required blocks can be found in the sounds block category.

8.1 Control your sounds

What makes a difference in a game, animation or story? Exactly music and sounds. Not sure? Watch a horror movie with sound and without sound and see the big difference this makes.

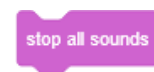
First, we must add our sound that we want to play later to the scratch project by using the sound editor. Select a sound from the scratch library, from your computer or record one directly in scratch. After this give a meaningful name to the sound and edit the sound if required. Thereafter you can use the **play sound () until done** stack block where the argument is the name of the sound from the dropdown or any reporter block. The block will, when executed, start playing the sound and pausing the script. The block **start sound ()** starts the sound without pausing the script. When the **start sound ()** block is executed again with the same sound as already running, the sound will be cut off and started again. If it is executed with a different sound, both sounds are played. To stop the sounds, use the stack block **stop all sounds**.



Play sound () until done stack block

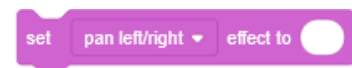


Start sound () stack block

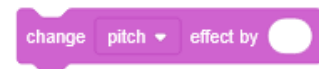


Stop all sounds stack block

As for costumes scratch also provides sound effects. By using the stack blocks **change () effect by ()** and **set () effect to ()** from the sound block category it is possible to apply sound effects. As the blocks from the looks block category do have the same name you might think they also work the same way but there is a slight difference. The effects specified with the looks block category apply to one single sprite or costume, the blocks from the sounds block category do apply to all sounds currently playing as it is not possible to specify to what sound the effect should be applied. The usage of the blocks remains equal where the first argument is the sound effect selected from the dropdown list and the second argument is the amount you want to change or set to. Scratch provides the following sound effects:



Set () effect to stack block

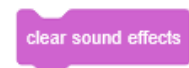


Change () effect by () stack block

Pitch: The pitch effect wraps the pitch upwards or downwards. A change of 10 corresponds to one half-step either upwards or downwards. To change a whole octave twelve, halve steps are used corresponding in a change of 120. Use negative numbers to move downwards and positive numbers to move upwards but stay in the range from -360 to 360. Values outside the range will be interpreted as the border value.

Pan Left/right: The pan effect causes the audio to shift towards the left or right output track depending on the value. A positive value shifts the audio to the right, a negative value shifts the audio to the left. Use a value between -100 and 100. Values outside the range will be interpreted as the border value.

You can use both sound effects on the same sound at the same time. To reset all sound effect scratch provides the **clear sound effects** stack block. Scratch also provides some other sound effects such as faster, slower, louder, softer, fade in, fade out, reverb or robot inside the sound editor. Be aware that those sound effects permanently apply to the sound, and it is not possible to change or reset them during the game, animation or story.



Clear sound effects stack block

An important part of sounds is the volume. The current volume is stored in the **volume** reporter block. The value of this block is between 0 (minimum) and 100 (maximum). The volume is globally for all sounds that will or are playing. For the same reason as for sound effects it is not possible to set a custom volume for a single sound, but the sound editor provides the louder and softer filter to adjust volume of a single sound. To adjust the volume



Volume reporter block

of all sounds, use the **change volume by ()** and **set volume to () %** stack blocks. If you use a positive number in the **change volume by ()** stack block argument the volume will increase, if you use a negative number it will decrease as the value is added to the current volume what is available in the **volume** reporter block. If the resulting value is outside the volume range the corresponding border value is used. For the **set volume to ()** stack blocks argument use a number in the range of the volume from 0 to 100. For both stack blocks it is possible to use a reporter block representing a value inside the volume range as an argument.



Change volume by () stack block



Set volume () % stack block

8.2 Exercises

Exercise 8_01: Play the sound cave from the scratch library infinitely when start is pressed.

Exercise 8_02: Play the sound Jungle from the scratch library when start is pressed and change pitch by two octaves after 1.5 seconds.

Exercise 8_03: Play the sound Dance Energetic and Drum Machine from the scratch library infinitely at the same time when start is pressed

Exercise 8_04: Play the sound Movie 2 and Mystery from the scratch library infinitely at the same time when start is pressed. Movie 2 should be louder than Mystery.

8.3 Question space

9 Sensing

Previously I talked about the 5 senses of a human but not only human have senses as well our sprites do have senses. This chapter will enable you to get used to the sensing of your sprite. All required blocks can be found in the sensing block category.

9.1 Touching something?

You have learnt a lot about aesthetic options in scratch, let's go back to programming and talk about touching. To check if something does touch something else you can use operators block and the Pythagoras to calculate the distance between two points, but scratch provides an even better solution with the operator block **distance to ()**. This block calculates the distance between your sprites center and the specified other object. You can select in the dropdown menu the mouse pointer or any other sprites center as well you can use a reporter block. If you use an invalid value for the argument the value 10'000 will be reported. This block does work independently if the sprite is hidden or shown.



*Distance to ()
reporter block*

Some advanced calculations with the distance can tell you if the two objects are touching each other. As this is a functionality often used in scratch, scratch provides a boolean block **touching ()?** to find out if the objects are touching. Valid values can be found in the dropdown menu and are the mouse pointer, edge or any other sprite. The block will return true if the visual texture of the sprite touches the specified object. This means that when your sprite is a circle it will return true when the border or the circle touches the other object and not when the rectangular frame touches the other object. The block will return false when the sprite does not touch the object and if the block is compared to another sprite that is hidden. If you use the filter ghost, the sprite will not be visible as with hidden, but the block **touching ()?** would return true as the sprite is still detectable.



Touching ()? boolean block

Sometimes a sprite should move over a border defined by a color. For this problem scratch provides the boolean block **touching color ()?**. This block returns true when the sprite does touch the specified color else false. The block is much slower than the **touching ()?** block and therefore is not suited to detect if two sprits touch each other. As well this block returns often positive false results because the stage ca display more than 16 million colours whereas this sensing block can differentiate only a much smaller number of colours. Normally you will not notice this, only if you need absolute precision.



*Touching color ()?
Boolean block*

If you need even more detailed result, you can use the sensing boolean block **color () is touching ()?** block. This block does check if the first specified color inside the sprite does touch the second specified color. This block must deal with the same limitations as the **touching color ()?** block and faces the same performance issues.



Color () is touching ()? boolean block

9.2 Ask the player

In scratch you can inform the user about something by using the looks block **say ()** or **think ()** but maybe you want to receive some information from the user such as name, age or an answer to a question. Scratch provides the stack block **ask () and wait**. This block will create an input box at the bottom of the stage. The first argument of the block contains the question provided to the user. The given answer to the question is stored in the **answer** reporter block. The value of this block changes after the user entered a new answer. Store the answer in a variable if you need it again or handle the action directly.



Ask () and wait stack block



Answer reporter block

9.3 Key pressed and mouse movement

In previous chapters we learnt how to start a script if a key is pressed. The sensing block category provides the boolean block **key () pressed?** to check whether a specified key is pressed. By default, you can select all letters from the English alphabet. If you need more specialised letters, use a reporter block containing a string with



Key () pressed? Boolean block

your letter. This block can be helpful to structure your code and to minimize the number of scripts. The block is also very useful if you require multiple keys pressed to execute an action.

If you want to check if any of the computer mouse buttons are pressed while the cursor is over the stage use the boolean block **mouse down?**. With the help of the two reporter blocks **mouse x** and **mouse y** you can get the position of the mouse while mouse down. This combination of blocks helps you to detect if a sprite is clicked by using the x and y coordinates of the sprite or for dragging an object.



Mouse down?
boolean block



Mouse x and y reporter
block

If you want to simply drag and drop the sprite you must set the drag and drop mode of the sprite with the stack block **set drag mode ()** to draggable if you do not want it to be draggable use the not draggable option. This effect does only work in the full screen mode of your project. You can use this block as well in a condition what allows the sprite to be draggable if a certain key is pressed or the game is in a certain state depending on a variable, else not.



Set drag mode () stack block

9.4 Loudness of the environment

You are tired of the normal control options with arrows or letters and want something innovative to control your sprites? Scratch provides the reporter block **loudness**. This block requires a connected microphone and measures the loudness of the environment. If no microphone is connected the reporter block will return -1. The value of the block goes from 0 (silent) to 100 (loud) and measures how quiet the microphones input is. If the sound would be louder than 100 only 100 is returned. But how to control your sprite now? For example, the louder a sound is the faster the sprite moves or the higher it jumps. Maybe you will find even more interesting implementations of this tiny but powerful reporter block.



Loudness reporter block

9.5 Control the time

Some projects need to measure the time a user needs to complete a task or how long the user survives therefore scratch provides a timer. The timer counts the seconds from when last time the green flag was clicked or the timer was reset using the **reset timer** stack block. The value of the timer is updated once per frame, so each 1/30th of a second. The value of the timer can be retrieved with the **timer** reporter block.



Timer related blocks

Sometimes it is also important to know what day what is today is possible with the reporter block **days since 2000**. This block returns a decimal value of days and its fractions since 1st January 2000 (UTC). If you want to get the current year, month, date, day of week, hour, minute or second scratch provides the **current ()** reporter block. This block only allows argument values from the dropdown menu and reports the selected date value. This makes it very simple to get the current date.



Time related reporter blocks

9.6 Get values of other objects

A powerful block provided by scratch is the **() of ()** reporter block. This blocks first argument takes a dropdown option of ether x position, y position, direction, costume #, costume name, size or volume for sprites and backdrop #, backdrop name or volume for backdrops. Sprites and backdrop also include local variables as options. The second argument does specify a stage or a sprite. The block will report the value stored in this variable of the selected stage or sprite. This can be useful for position comparisons (same location) or game state of a sprite.



() of () reporter block

9.7 Exercises

Exercise 9_01: Ask the name of the user and wait for the response. Say hello to the user after retrieving his name.

Exercise 9_02: Move with your sprite to the right and stop moving when touching the border.

Exercise 9_03: Control the movements of the sprite with the arrow keys. Use one single script to complete this task (do not use the event hat block when () key pressed).

Exercise 9_04: Create a bumping heart where the beats intensity is depending on the surrounding loudness.

Exercise 9_05: Show the current date in the following format Year-month-day
hours:minute:second.

9.8 Question space

Appendix I – Cheat Sheet

Rotation

The sprite can rotate 360 degrees. But scratch sometimes you see the same value for different directions only differentiate by the sign. This is because the rotation in scratch is clockwise. Imagine you are at twelve. You can either move 270 degrees clockwise or 90 degrees counterclockwise. Scratch chooses the shortest rotation.



Direction -90 or 270 degrees

Direction 90 or -270 degrees

Rotation Style

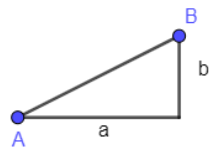
Scratch provides three rotation styles:

All Around: Sprite pointing in given direction

Left-right: Sprite pointing to the right if given value is between 0 and 180 otherwise left.

Don't Rotate: Does always point to the right or left independent of point direction

Distance between two points



To calculate the distance between two sprites, use the built-in reporter block **distance to ()** located in the sensing block category. If you need to calculate the distance between two

points use the Pythagorean theorem following the instructions below.

1. Calculate the distance of a and b by using the following scratch code



2. Calculate the distance by using the Pythagorean theorem using the following scratch code



Touching border?

The stage of scratch is 480px wide and 360px tall. Use the built-in boolean block **touching ()?** located in the sensing block category to check if the sprite touches the border or any other sprite.

To check if the sprite touches another border check whether the sprite position is less than or greater than the min or max value on each axis. The sprites coordinates are always located at the center of the sprite. Add or subtract half of the sprites size to check whether the border was touched or not.

Messaging

Use messages to notify all other sprites. Each message can be sent using the **broadcast ()** stack block. If you use **broadcast () and wait** the sending script pauses until all scripts invoked by the messages are finished. Use **when I receive ()** hat block to launch a script when a specific message is received.

Sources

Information and Medias from the official Scratch Wiki or self-made.
https://en.scratch-wiki.info/wiki/Scratch_3.0