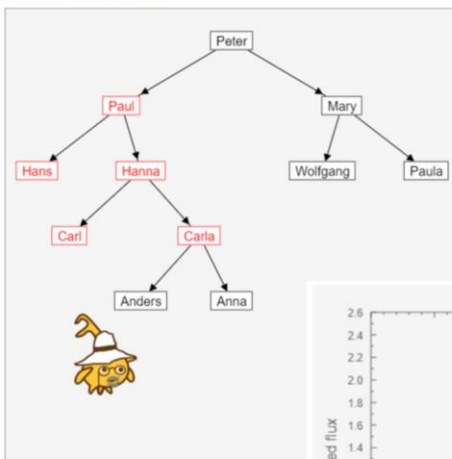
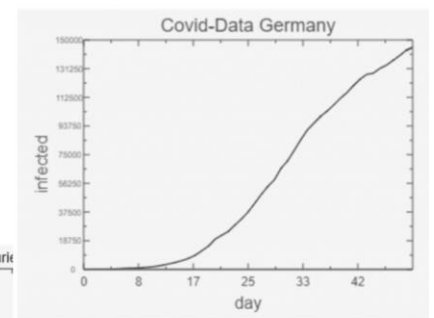
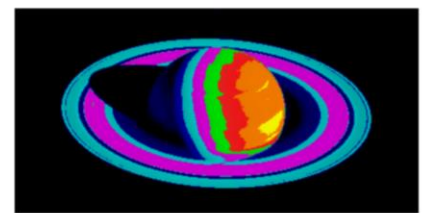
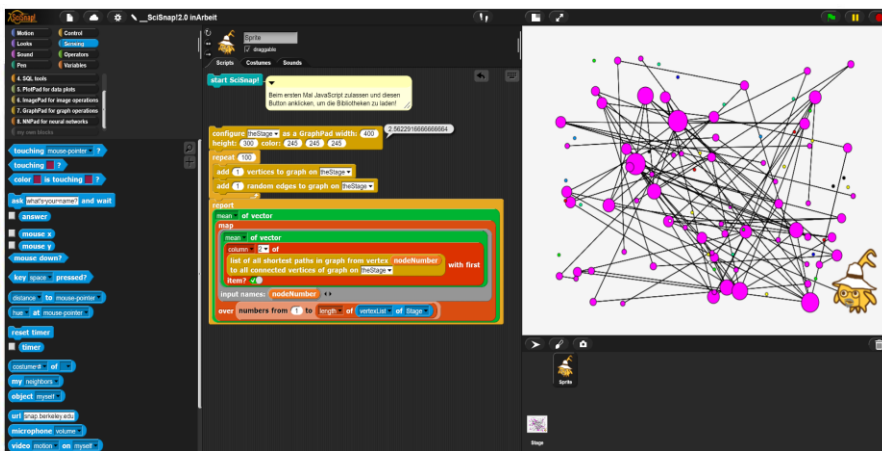
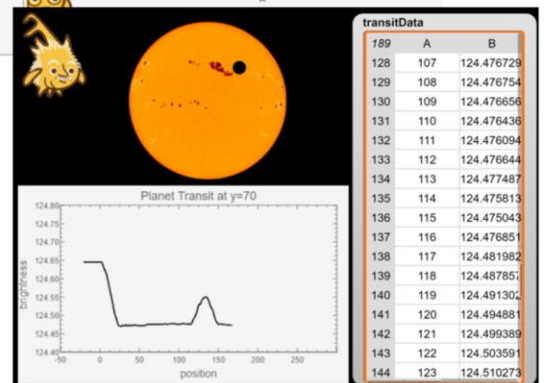
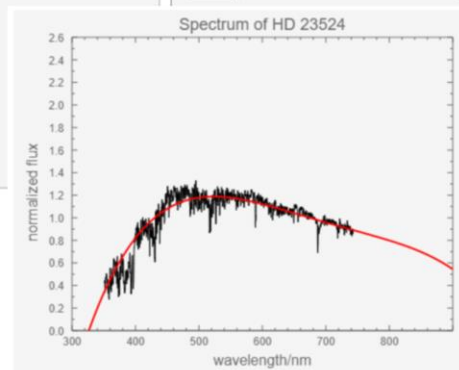
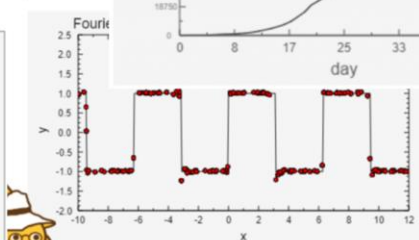
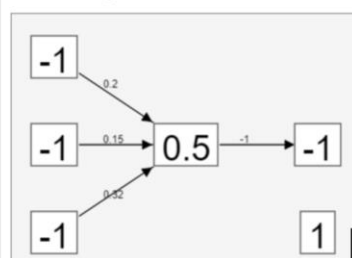


Eckart Modrow

Programming with version 2



Click on sprite to learn!



© Eckart Modrow 2022

emodrow@informatik.uni-goettingen.de



This work is licensed under a Creative Commons attribution - Non-Commercial - Share Alike 4.0 International License. It allows download and redistribution of the complete work with mention of my name, but no editing or commercial use. The scripts are developed with *Snap! 7.3.1 Build Your Own Blocks*.

The examples and this script can be loaded at

<http://emu-online.de/SciSnap2Examples.zip> or
<http://emu-online.de/ProgrammingWithSciSnap2.pdf>

SciSnap!2 itself at

<https://snap.berkeley.edu/snap/snap.html#present:Username=emodrow&ProjectName=SciSnap!2&editMode>

Prof. Dr. Modrow, Eckart:

Programming with *SciSnap!2*

© emu-online Scheden 2022

All rights reserved

If this book is helpful for you and you would like to express your appreciation in form of a donation, you can do so at the following PayPal account:

emodrow@emu-online.de
Purpose: SciSnap!-book



This publication and its parts are protected by copyright. Any use in others than legally permitted cases requires the prior written consent of the author.

The software and hardware names used in this book as well as the brand names of the respective companies are generally subject to the protection of goods, trademarks, and patents. The product names used are protected by trademark law for the respective copyright holders and cannot be freely used.

This book expresses views and opinions of the author. No guarantee is given for the correct executability of the given sample source texts in this book. I assume no liability or legal responsibility for any damages resulting from the use of the source texts of this book or other incorrect information.

Preface

The development of computer tools, especially in the field of programming languages, has made rapid progress in recent decades. For example, graphical programming languages have been developed that allow beginners to work on small projects on their own very quickly without having to worry about syntax quirks, etc. If only a limited time is available for learning programming, this is a decisive step forward, because the relationship between the practice of using the programming tool (the programming environment incl. language) and the content work practically reverses. Accordingly, tools like *Scratch*¹ from *MIT* or *Snap!*² from *UCB* are used successfully in schools and universities.

So, although a lot has happened with the tools, the content in programming courses looks surprisingly unchanged. Simple "tasks" are set, which largely serve only to practice the dealing with algorithmic basic structures and data structures, without going beyond that. In addition, there are often working techniques that make sense for large projects with many participants, but which are hardly experienced as helpful by the beginners. An example may be the drawing of Nassi-Shneiderman diagrams³ (structograms), which sometimes have to be made before scripts are developed, e.g. in Scratch – even though graphical languages illustrate the algorithmic structure through their blocks themselves. That is exactly what they were developed for (among other things). One can imagine the enthusiasm of the learners, e.g. if they have to add some numbers and calculate the tax to be added or as a "funny interlude" to replace all "r" in a text with "l" and thus produce "Chinese" texts. Consequently, the "successes" in programming lessons are also largely unchanged. Because independent problem solving with the resulting product pride is as rare as meaningful applications that explain parts of the learners' world, often only those learners feel addressed who are "interested in computers" anyway. The others, i.e. most of them, also meet the requirements, but they rightly ask themselves: "Why should I learn this, what's for?"

The objection that you can only treat elementary examples with beginners is not to be dismissed. Although with graphical languages there is much more time for actual problem solving, the algorithms that are developed independently at this stage of learning are fairly simple. They usually involve a sequence of commands, often within a loop, that lists some alternatives in sequence: "If this is the case, then do so." If such scripts are nevertheless to provide meaningful experiences, then the elementary commands used – a few – must be "powerful," and it is necessary for the teachers to be imaginative in order to teach "interesting" problems at an elementary level.

This is not a new insight: In the days of the Nassi Shneiderman diagrams, it was a challenging task to draw an oblique line on a technical device such as a screen or printer. Since corresponding graphics commands were developed, it has been a trivial problem that is dealt with one instruction. Just a few years ago, measuring with computers was something for specialists. Today, almost every school has a set of sensor boards that children work with. It is actually hard to understand why the new possibilities hardly appear in the field of algorithmics. A few random numbers are still sorted or words are converted to capital letters instead of "digging" into sets of data using similarly simple scripts or searching books or images for characteristic structures. To be precise: if characteristic features such as means, standard deviations, ... grouped by characteristics such as the place of residence, gender or occupation of the parents can be determined in a data set with one command, there is enough space in the rest of the algorithm e.g. searching for correlations or graphing the relationships, also with one or a few

¹ <https://scratch.mit.edu/>

² <https://snap.berkeley.edu/>

³ This type of diagram was developed in 1972. For chronological classification: one year later, the first microprocessor was launched with the Intel 4004. This may speak for the timeless importance of structograms, but it may also indicate that changes could be considered after 50 years.

commands. Above all, however, these possibilities can raise current and obviously important questions, the answers of which concern the learners themselves.

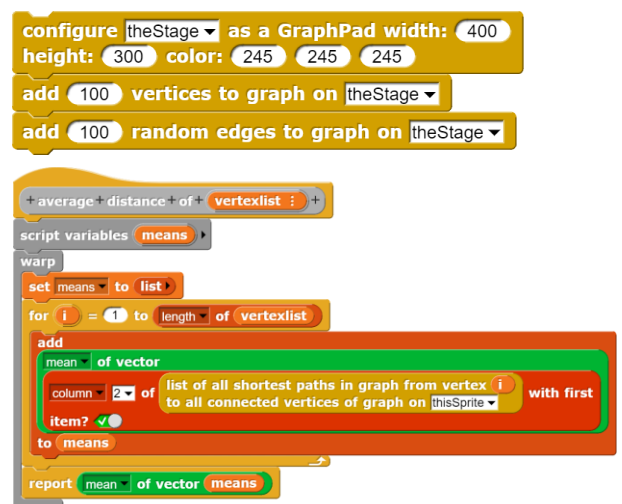
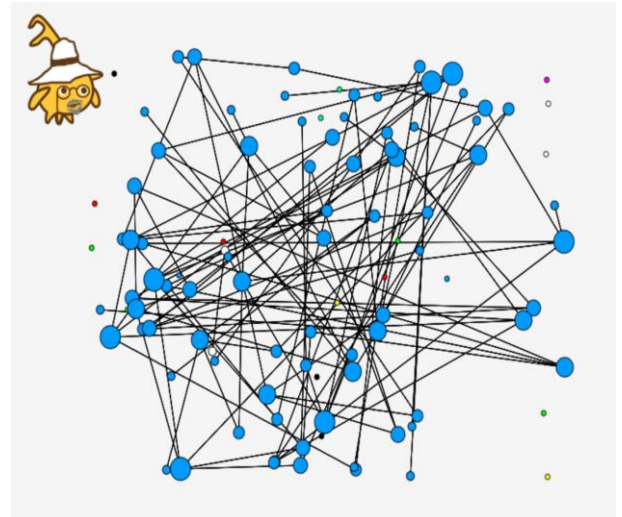
1. A goal of *SciSnap!* is therefore to provide appropriate libraries in various fields such as image processing, diagramming, mathematics, data analysis and databases, graphs or neural networks.

A (unfortunately) current example: economic, geographical, social or content relationships can be represented by graphs. If appropriate commands are available, then the creation and representation of such a (here: random) graph in *SciSnap!* requires only three commands: "configure a sprite, create n nodes, and then n randomly selected edges"⁴. If we consider the links as contacts between persons, then the question is how many intermediate contacts can spread infections during pandemic times. Thus, we compute the shortest paths between the nodes: "For each node: compute the shortest paths to all other nodes and enter them in a list". For these results, we calculate the mean values per node in a simple loop, and from this we calculate the overall mean value.⁵ Algorithmically, it's a typical beginner's problem: "run through a simple loop". In terms of content, we have found ways to discuss a current social problem, "small worlds"⁶, social networks, friendships or customer-supplier relationships. Teaching has become more relevant.

Starting with *Snap!7.0* there is the possibility to create or delete additional palettes. This significantly increases the overview. Nobody will need all of the eight partly extensive libraries of *SciSnap!2* as well as the category *my own blocks* at the same time. Nevertheless, I did not split them up further, because you can switch off the ones you don't need with one click (*Remove a category...* from the file menu). However, the first three (*SciSnap! globals*, *Math tools* and *Data tools*) as well as some new blocks in the standard palettes should not be omitted.

It's not the goal of *SciSnap!* to present ready-made applications. Rather, it provides powerful commands that can be used to build applications. An example of this are the "Sketchpads": costumes for arbitrary sprites or the stage, on which sketches can be created quickly. Function graphs, images, charts, or histograms can be created with a few commands, and scales are added—and deleted when it gets too crowded. This makes it possible, for example, to illustrate mathematical relationships, such as showing the effect of operators on complex numbers. It is hoped that these examples will encourage learners to create even other, perhaps better, applications that use algorithmic methods in different fields.

2. *SciSnap!* should be usable both as a tool and as a development environment.



⁴ Pseudocode: *configure GraphPad, add 100 vertices, add 100 edges*

⁵ Pseudocode: *means=list, for i=1 to 100(add mean of row i of distances to means), mean=mean of means*

⁶ https://en.wikipedia.org/wiki/Small-world_experiment

Snap! is not only a fantastic development tool, but it is based on a fantastic concept. As a graphical reimplementa-
tion of MIT's *Scheme*⁷ language, based on the "CS Bible" "*Structure and Interpretation of Computer Pro-
grams*"⁸ by Abelson et.al., it is conceptually far superior to many of the most common programming languages.
Although it's not very fast, *Snap!* is running fast enough to be used fluidly in educational settings. Its built-in
visualization capabilities make it ideal for simulations. The prototypical inheritance used makes basic computer
concepts directly tangible. Nevertheless, it is largely underrated, probably because of the similarity of its interface
to *Scratch*. I hope, therefore, that making libraries available that are intended more for projects in high school or
in the first semesters of college will have a positive effect on its distribution to these age groups. Let's see...

3. *SciSnap!* is intended for higher grades of school as well as for undergraduate study.

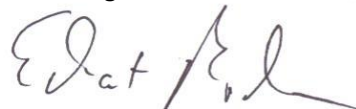
This script contains a description of the possibilities of *SciSnap!* as well as some examples explaining the intended
use. The libraries are based on the experience with "*Machine learning with Arthur&Ina*"⁹ and the *Snap!*-fork
*SQL-Snap!*¹⁰. They were supplemented by numerous mathematical operators, SQL, sketchpads, neural net-
works and graphs. A detailed description of *Snap!* with a lot of examples can be found at "*Computer Science with
Snap!*"¹¹ – and of course, in the *Snap!*-manual¹². The presented concepts have been and will be used in classes
and in beginner lectures at the university.

Oh, and of course there are also two little helpers that will assist in your work with
SciSnap!. Depending on the application, they two will take turns. *Alberto*¹³ will take care
of the more scientific applications, *Hilberto*¹⁴ of mathematical and data-oriented ones. If
one is active, the other can rest a bit. Both claims to be distant relatives of *Alonzo*, the
Snap!-Mascot. Whether that's true? One does not know!

I would like to thank Jens Mönig and especially Rick Hessman for his contributions to the
PlotPad, their support and the numerous discussions and suggestions. And many thanks
for the wonderful help of the *DeepL*¹⁵ translation program. I would probably never have
finished without it.

I hope you enjoy working with *SciSnap!*.

Göttingen, on 2022.3.22




⁷ https://en.wikipedia.org/wiki/MIT/GNU_Scheme

⁸ <https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book.html>

⁹ as well as other materials on <http://emu-online.de>

¹⁰ <http://snapextensions.uni-goettingen.de/>

¹¹ <http://ddi-mod.uni-goettingen.de/InformatikMitSnap.pdf>

¹² <https://snap.berkeley.edu/snap/help/SnapManual.pdf>

¹³ he works in astrophysics with Rick Hessman

¹⁴ https://de.wikipedia.org/wiki/David_Hilbert

¹⁵ <https://www.deepl.com/translator>

Content

Preface	3
Content	6
1 Starting <i>SciSnap!</i>	9
2 New Blocks in the Standard Palettes	10
3 The Structure of <i>SciSnap!</i> -Sprites	12
4 The <i>SciSnap!</i> -Libraries	14
4.1 Blocks of the SciSnap! globals palette	14
4.2 The Math Library	16
4.2.1 Linear Algebra	16
4.2.2 Complex Numbers	18
4.2.3 The MathPad	19
4.2.4 Numerical Methods	20
4.2.5 Statistics	21
4.2.6 Sets	22
4.3 The Data Library	24
4.4 The SQL Library	29
4.5 The PlotPad Library	32
4.5 The ImagePad Library	34
4.6 The GraphPad Library	37
4.7 The NeuralNetPad Library	39
5 Data Import and Export	41
6 Math related Examples	45
6.1 Representation of complex numbers	45
6.2 Affine transformation of a triangle in \mathbb{R}^2	46
6.3 Rotation of a pyramide in \mathbb{R}^3	47
6.4 Graph of normal distribution	48
6.5 Cartesian product of three sets	49
6.6 Representation of a set of points and the regression line	50
6.7 Interpolation polynomial through n points	51
6.8 Approximation of a tangent by secants	53
6.9 Finite series	55
6.10 Application of the Taylor series to the mathematical pendulum	57
6.11 Fourier expansion for a square wave signal with numerical integration	60
6.12 Drawing a function and its derivatives	64
6.13 Random experiments on the binomial distribution	65
6.14 Fast Fourier Transform (FFT)	67
6.15 A simple image compression method with FFT	71
6.16 A simple sound compression method with FFT	74

7	Data related Examples	75
7.1	Data plot of points scattering around a function graph	75
7.2	Histogram of random values	76
7.3	Plot of mixed data	77
7.4	NY Citibike Tripdata 1: Correlations	78
7.5	NY Citibike Tripdata 2: Usage of bikes	79
7.6	NY Citibike Tripdata 3: World Map Library	80
7.7	NY Citibike Tripdata 4: Lending diagrams	81
7.8	Income data from the US Census Income Dataset	84
7.9	Covid-19-Data Analysis	86
7.10	Star spectra	88
7.11	Flu wave simulation	91
8	Graphic related Examples	93
8.1	Simple random graphic	93
8.2	False color image of a lunar crater	94
8.3	Slice through the image of the lunar crater Tycho	94
8.4	Shadow lengths in the lunar crater Tycho	95
8.5	Plot of image data as histogram	96
8.6	Simulation of a planetary transit in front of a sun	97
8.7	Affine transformation of an image	99
8.8	Kernel applications for edge detection in images	100
8.9	Diffusion in the grid model	101
8.10	Ferromagnetism as a Grid Automaton	102
8.11	Conway's Game of Life	103
8.12	A cellular automaton as a soft focus device	105
8.13	Linear Wolfram automata	107
9	SQL Examples	108
9.1	Working with the SQL library	108
9.2	Simple SQL query	109
9.3	More complex SQL query	109
10	Graph Examples	110
10.1	Mean distances in a random graph (small worlds)	110
10.2	Mean distances in a scalefree graph (small worlds)	110
10.3	Edges per vertex histogram for a random graph	111
10.4	Edges per vertex histogram for a scalefree graph	111
10.5	Breadth and depth first search in a family tree	112
10.6	A graph lab	114
10.7	Tree search in the graph lab	115
10.8	Graph lab with world map library	116

11 Machine Learning Examples	117
11.1 A simple perceptron as a graph	117
11.2 A simple learning perceptron as a graph	119
11.3 Training of a neural network	121
11.4 Traffic sign recognition with a neural network of perceptrons	122
11.5 Under- and Overfitting	127
11.6 Classification in the HR-diagram according to the kNN method	130
11.7 Decision trees according to the ID3 method	132
11.8 K-means-clustering	134
11.9 Clustering according to the DBSCAN method	137
11.10 Outlier detection according to the DBSCAN method	139
11.11 DNA-Clustering with Levenshtein distance	140
11.12 Character recognition with a Convolutional Neural Network	141
11.13 Reinforcement learning / Q-learning	146
 Notes	 149
References and sources	150

1 Starting SciSnap!

SciSnap! consists of a collection of quite normal *Snap!* blocks, arranged in eight palettes, which differ in their functionality. Another palette *My own blocks* is for your own programs. In addition, there is a sprite named *Hilberto* with a few costumes, but it is not necessary for the blocks to work. Of course, nobody needs all palettes at the same time; nevertheless I left them in one package, because from *Snap!7.0* on it is possible with a few clicks to delete the not needed parts quickly and to save the changed configuration as a new working environment.

The first four palettes contains blocks that can be used directly. Except for the SQL commands, they should not be deleted. The commands of the other four palettes each refer to sprites that have been configured with the first block of the palette - to *SketchPads* for diagrams, images, graphs, or neural networks. These palettes can be deleted depending on the needs in the project.

The easiest way to start *SciSnap!* is to simply call the corresponding link.

<https://snap.berkeley.edu/snap/snap.html#present:Username=emodrow&ProjectName=SciSnap!2.0&editMode>

Of course, the *SciSnap!* blocks can be loaded instead - but in this case without *Hilberto*. The second is recommended if an existing project is to be extended with *SciSnap!* functionalities.

SciSnap! works with a set of JavaScript libraries stored on a server. They are loaded automatically by the block *start SciSnap!* before *Snap!* is reconfigured to *SciSnap!* which can be recognized e.g. by the changed logo. You can find *SciSnap!* in the *Snap!* libraries in the File menu. If you load the blocks from another server, then you must explicitly allow access to JavaScript extensions in the settings menu in case *Snap!* does not trust the server.

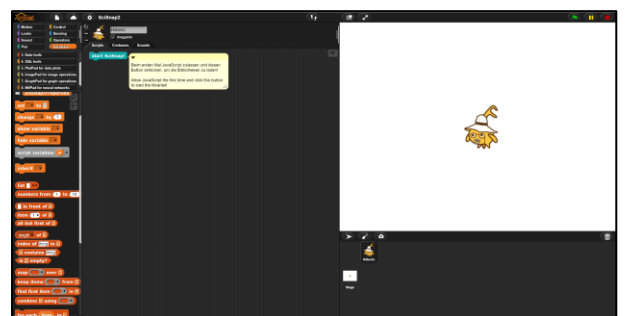
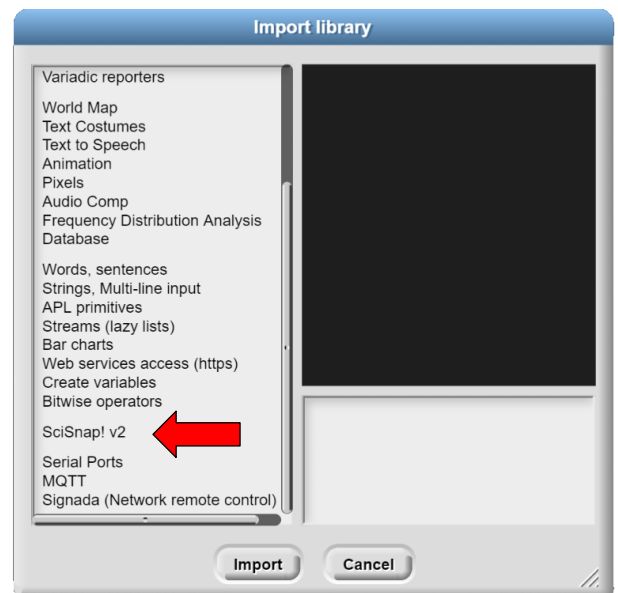
We will go through this with an example: The goal is to work with the SQL blocks and the palette for custom blocks. It should also be possible to create diagrams.

1. Step: Either we load *SciSnap!* as a normal *Snap!* library (without *Hilberto*) or we load *SciSnap!* from the *Snap!* cloud

<https://snap.berkeley.edu/snap/snap.html#present:Username=emodrow&ProjectName=SciSnap!2&editMode>
















(as a project with *Hilberto*). We get the adjacent screen. There we enable JavaScript as needed.















2. Step: We delete the last three *SciSnap!* palettes using the file menu (*Remove a category...*). **In any case we click on the *start SciSnap!* button.** As a result, we get a SQL working environment, in the picture with the result of a first query.



2 New Blocks in the Standard Palettes

Some blocks of *SciSnap!* are located in the standard palettes because they logically belong there. They mostly complement the functionality that is already there anyway. These are the following blocks:

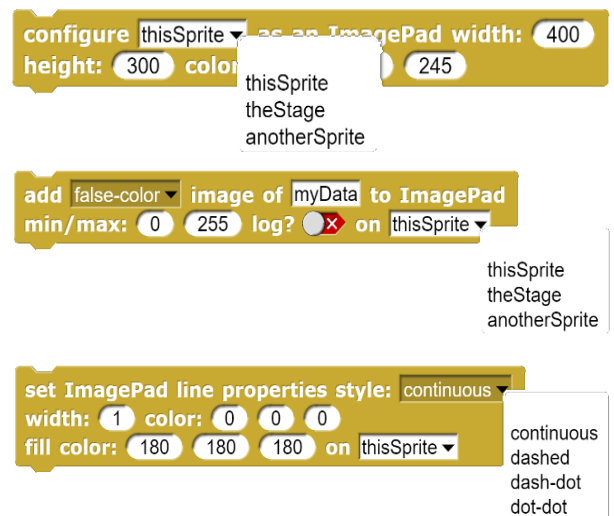
 	Provides the costume of a sprite, for example for pooling operations.
 	Provides a copy of a costume.
	Provides a costume of the specified size and color.
 <div data-bbox="592 797 719 875"> thisSprite theStage anotherSprite </div>	Adds a copy of the current costume to the costume list of the sprite or stage.
	Imports a sprite that was exported as an XML file.
	Deletes the calling sprite.
 	A block from the <i>Snap!</i> libraries: Replaces a nested block sequence with a somewhat clearer structure.
 	Date and time in standard format, e.g. for astronomical purposes.
 <div data-bbox="616 1391 799 1637"> Julian Date decimal years days this year hours this year minutes this year seconds this year hours today minutes today seconds today </div>	Provides date and times based on the standard format.
  <div data-bbox="215 1827 699 2029"> Dateneingabe der Farbe: (click on it) 1: Rot 2: Grün 3: Blau 0 ok </div>	Simple input dialog by selecting the desired answer with the mouse.

	Generates random numbers between 0 and 1.
	Returns π .
	Returns Euler's number.
	Rounds a number to the specified number of decimal digits.
	Returns the factorial of a natural number.
	Calculates a binomial coefficient.
	Returns the specified part of a string.
	Deletes a substring in another, either in all places or the first one.
	Converts a string to uppercase.
	Converts a string to lowercase.
	Saves a text in the specified file in the download directory of the browser.
	Returns the position of the first character of the first occurrence of the specified substring in a string.
	Replaces one substring in another, either in all places or the first one.
	Creates a label for coordinate system axes from a column with texts.

3 The Structure of SciSnap!-Sprites

The first part of *SciSnap!* consists of four libraries that are generally applicable in *Snap!* scripts (*SciSnap! globals*, *Math tools*, *Data tools*, *SQL tools*). The other four libraries work with specially configured sprites (*PlotPad*, *ImagePad*, *GraphPad*, *NNPad*). The reason for this is that the properties for a *NeuralNet* are very different from those for a *PlotPad*. In addition, such "special sprites" must have their own data areas in which, for example, image data can be stored. A global data area can be found in the *SciSnap!Data* variable, which e.g. the blocks of the *Data tools* library use in the default case. For the creation of diagrams, on the other hand, it makes more sense for a *PlotPad* to have its own data area that is independent of, for example, that of the *ImagePad* to which the diagrams refer.

Each sprite and the stage can be configured as "special sprites", e.g. as *ImagePad*. The local variables *myProperties* and *myData* are created and some useful presets are made for the properties. The properties are usually grouped into groups, e.g. costume properties (*costume-Properties*) or the way to draw lines (*lineProperties*). All blocks that require a specific configuration initially check the *typeOfConfiguration* property of the sprite they are to work with. If this is not correct, an error message is displayed. The groups of properties can be changed with the corresponding blocks.



The structure of the *SciSnap!* sprites is thus based on the idea of documented data sets consisting of two parts: the metadata describing the structure and content context of the data (e.g. number format, image dimensions, recording device, recording date, ...) and the associated pure data segments. Metadata usually consist of dictionaries - names with assigned values (e.g., "recording date: 2018/12/24"). Examples of this structure are *FITS* files [FITS], which are standard in astrophysics but are also used in the Vatican Library, or *JPEG* images from cell phones. Here, too, there are metadata (image size, degree of compression, date taken, ...), without which image generation would not be possible.

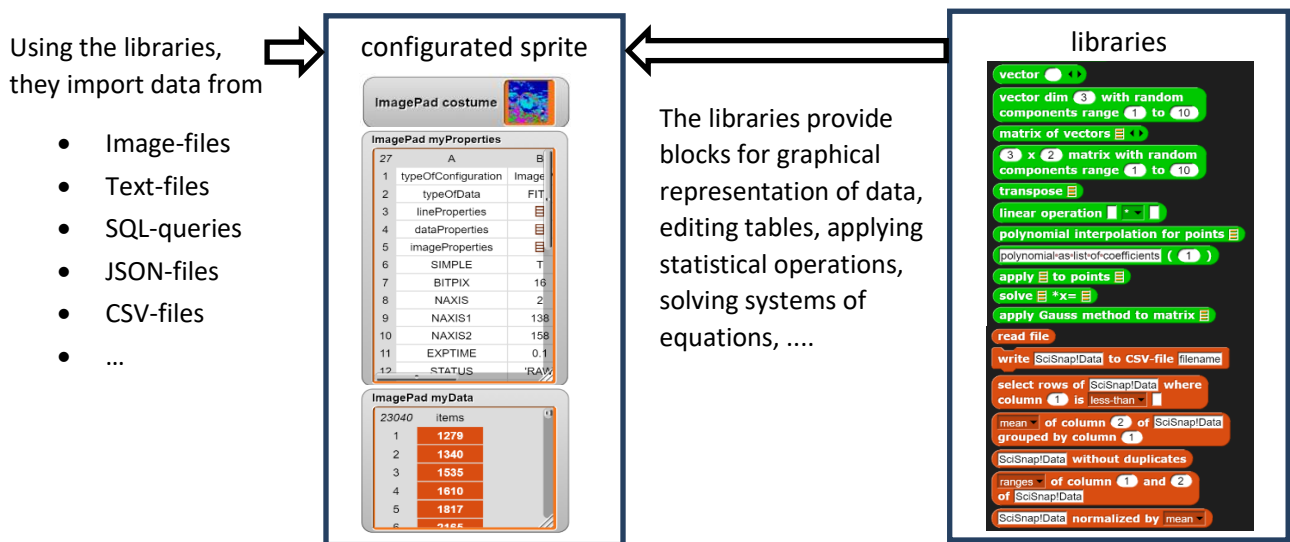
We adapt this structure by assigning two local variables to a *SciSnap!* sprite, each containing the data (*myData*) and the data description (*myProperties*). These variables can be filled by importing data from different sources (SQL query, text file, CVS file, JSON file, FITS file, direct assignment, ...), whereby the properties *myProperties* are to be adapted to the respective data. On the other hand, this can also be done "by hand". With the help of these properties, data can be converted into graphical representations (graph, data plot, histogram, image, ...), whereby either *myData* or another suitable table is selected as source.

It is important that the image generation does not change the original data. If, for example, an image of Jupiter is used to determine the distances of its moons, then these must at least be visible in the image. For this purpose, a false color image can be generated after setting some parameters. In this image Jupiter itself will appear rather unstructured. If, on the other hand, one wants to examine the "eye" of the planet more closely, then the parameters must be chosen quite differently, so that the moons are barely visible. All these changes must be done in the pixels of the current costume of the *Snap!* sprite without affecting the image data itself.

Because tables can be represented very nicely in *Snap!*, this form of representation is not implemented additionally. Instead, the data type *table* is implemented with many of the common operations in the field of *data science* (table operations, correlation calculation, affine transformations, solving systems of linear equations, ...), which can handle sufficiently fast even larger amounts of data.

Since *SciSnap!* (currently) contains about 250 new blocks, they have been grouped according to their functionality and distributed to different libraries and sprite configurations: a *Math tools* libraries for different areas of mathematics (60 blocks), a *Data tools* library (38 blocks) for handling the actual data, an *ImagePad* for image processing (25 blocks), a *PlotPad* for graphical representations (27 blocks), a *NeuralNetPad* for perceptron networks (11 blocks), an *SQL* library for database queries (26 blocks) and a *GraphPad* for graph theory applications (35 blocks). In addition there are the already mentioned blocks in the standard palettes of *Snap!*. All blocks are global and contain the target of the operation (*thisSprite*, *theStage* or the name of another sprite). Object-oriented calls are therefore largely unnecessary.

The configured sprites have the following structure:



Most blocks get their parameters (image size, value ranges, colors, ...) from the dictionary *myProperties*. The preset properties allow to use blocks for creating graphics, diagrams, ... without too many parameters. If the values do not fit, the properties are changed either individually or in groups.


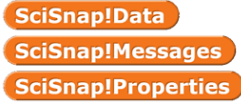





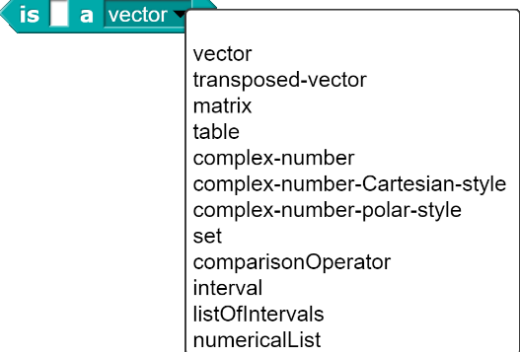
4 The SciSnap!-Libraries

In the following, the libraries are presented in tabular form. More extensive examples, which mostly use several libraries, follow afterwards.

The *SciSnap!* libraries - like all block libraries of *Snap!* - have been assigned to palettes and saved. If the palette of a block does not yet exist, then it is created during loading. If a library is to be loaded into another palette, then the block `import library to category Looks` can be used for this purpose. *My own blocks* is intended for the own blocks.

4.1 Blocks of the SciSnap! globals Palette

In this palette there are blocks for the configuration of *SciSnap!*.

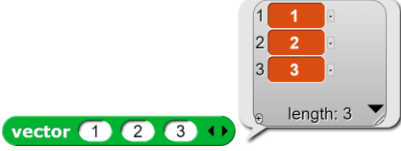
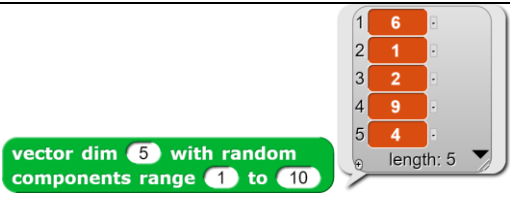


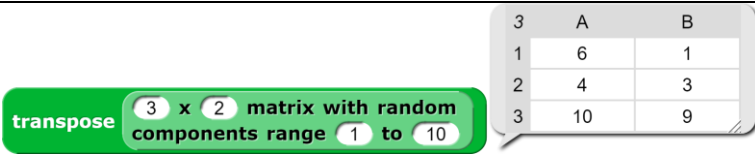
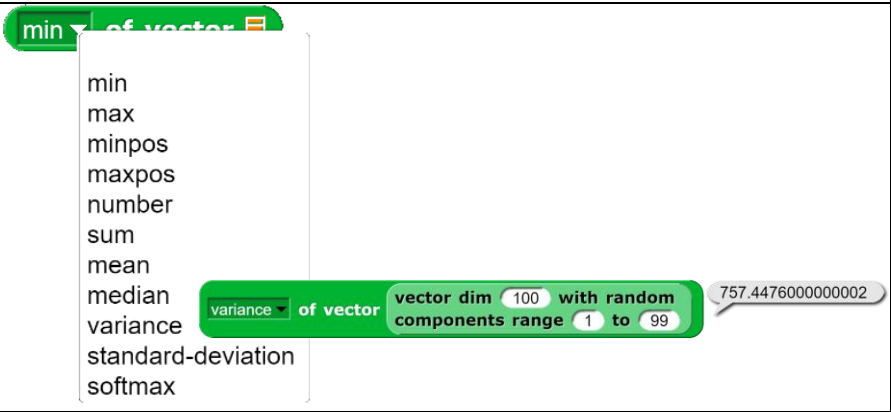
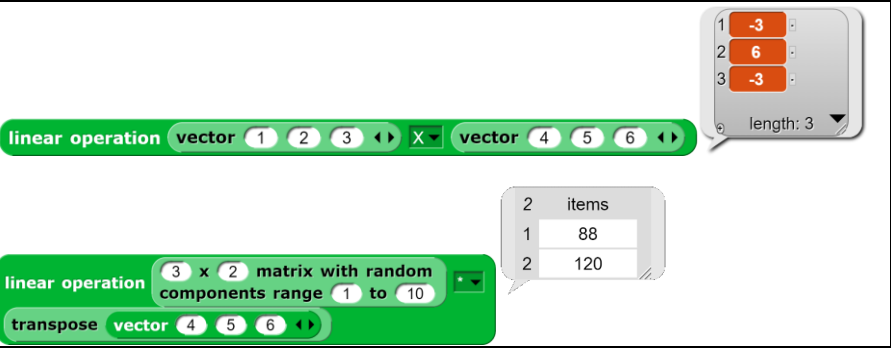

 	Loads the JavaScript library of <i>SciSnap!</i> . Creates the <i>SciSnap!</i> logo and the mapped global variables. Enlarges the stage to 800x600 pixels.
	Creates the global <i>SciSnap!</i> variables and sets some <i>SciSnap!</i> properties.
	Reads a property.
	Allows to change or to create a property.
	Creates a message window in the middle of the screen.
	Reports an error, if possible, via the current sprite and enters it into <i>SciSnap!Messages</i> .
	Tests an element to see if it belongs to a <i>SciSnap!</i> type.

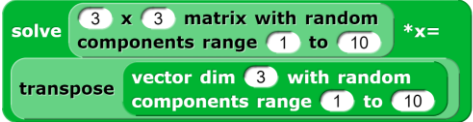
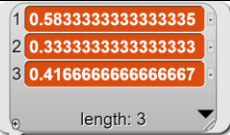

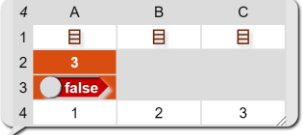

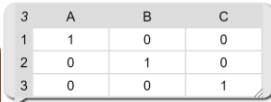


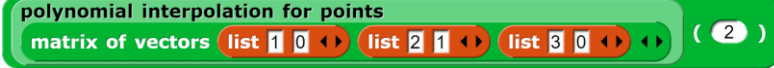

<p>is the global property <code>typeOfConfiguration</code> with value <code>MathPad</code> present ?</p>	<p>Tests whether a global or local property exists and whether it has the specified value.</p>
<p>Switch to SciSnap! logo</p>	<p>Replaces the <i>Snap!</i> logo with the <i>SciSnap!</i> logo.</p>
<p>import library to category Looks</p>	<p>Imports a library into the specified palette. All blocks should belong to the same category!</p>

4.2 The Math Library

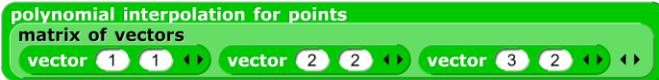

4.2.1 Linear Algebra


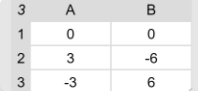
SciSnap! works with vectors and matrices and the common operations on them. Both are represented as lists or lists of lists, and both can be in transposed form. The following blocks work with them:

	Returns a vector of arbitrary dimension with given values.
	Returns a vector of arbitrary dimension with random values in the specified range.
	Returns the matrix from the specified vectors.
	Returns a matrix of arbitrary dimension with random values in the specified range.
	Matrices and vectors can be transposed.
	Returns one of the specified properties of a vector.
	Between scalars, vectors and matrices the assigned operations can be executed. Since vectors can be processed with the standard arithmetic operators of Snap! there are no separate blocks for them.
	Applies a matrix to a list of points (see example).

 	Calculates the solution to a system of linear equations.
   	The block returns several data of the passed matrix: the possibly diagonalized matrix, its rank, whether column swaps have occurred and the current order of the columns. If only the diagonalized matrix is of interest, then we look at the first element of the result.
 	Calculates the coefficients of the polynomial through n points.
	For polynomials you can calculate function values.
	Performs an affine transformation in the plane for a list of points, e.g. an image, described by the assignment of three points to three others (see example).

Examples:

4.2.2 Complex Numbers

If you are planning more extensive operations with complex numbers, you should consider using the Scheme library of *Snap!* (*Bignums*-library). *SciSnap!* is intended more for complex arithmetic as well as illustration of operations. In *SciSnap!* complex numbers are represented as 3-element lists, where the first entry denotes the format of the number: either the "Cartesian" style $z = a + b \cdot i$ or the polar form $z = r \cdot e^{i\varphi}$. There are input blocks for these two forms.

complex 2 + 3 * i	Returns a complex number in Cartesian form.
complex 2 * e ⁱ 30	Returns a complex number in polar form.

If necessary, these forms of representation can be converted into each other, and arithmetic operations can be performed.

complex Cartesian style	Returns a complex number in Cartesian form.
complex polar style	Returns a complex number in polar form.
complex complex 2 + 3 * i polar style	Example of format conversion.
complex real-part of	The components of a complex number can be accessed - regardless of their format.
absolute-value real-part imaginary-part phase conjugate	
complex +	And, of course, you can calculate with them.

Example for a multiplication:

complex **complex** 2 + 3 * i * **complex** 2 * eⁱ 30

4.2.3 The MathPad

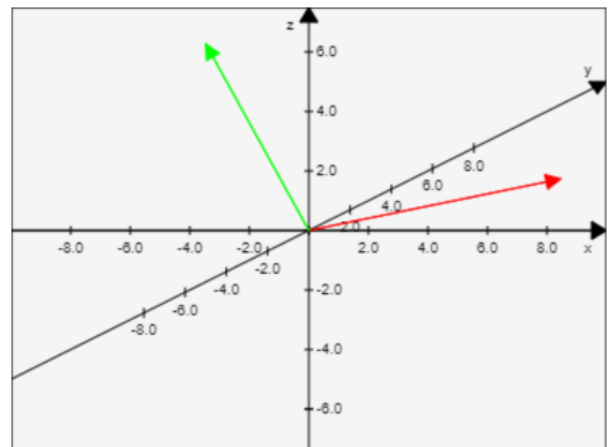
<pre>configure sprite thisSprite as a MathPad width: 400 height: 300 color: 245 245 245</pre>	Configures a sprite as a MathPad and draws a 3-dimensional coordinate system centered in the middle.
<pre>is thisSprite a MathPad?</pre>	Test on MathPad configuration.
<pre>set MathPadProperty costumeProperties of thisSprite to 1</pre>	Sets a MathPad property.
<pre>MathPadProperty costumeProperties of thisSprite</pre>	Reads a MathPad property.
<pre>set MathPad costume properties width: 400 height: 300 color: 245 245 245 offsets: 0 0 on thisSprite</pre>	Sets the costume properties of a MathPad to the specified values.
<pre>set MathPad properties lineWidth: 1 onlyPoints? <input checked="" type="checkbox"/> dimension: 3 maxValue: 10 startPoint: 0 0 0 on thisSprite</pre>	Sets the line properties of a MathPad to the specified values. If "onlyPoints" is set, only the endpoints are drawn.
<pre>add centered axes to a MathPad on thisSprite</pre>	Draws the coordinate system on a MathPad.
<pre>plot vector on MathPad vector 255 0 0 change startpoint? <input checked="" type="checkbox"/></pre> <div style="border: 1px solid black; padding: 2px; margin-top: 5px;"> vector complex-number line-to object-of </div>	Draws a vector, a complex number, a line or an object described by a list of vectors on a Math-Pad.

Example:

```
configure sprite thisSprite as a MathPad
width: 400 height: 300 color: 245 245 245

plot vector vector 5 5 0 color: 255 0 0
on MathPad thisSprite Change startpoint? ☒

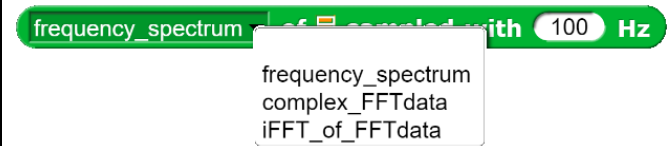
plot vector vector 0 -5 8 color: 0 255 0
on MathPad thisSprite Change startpoint? ☒
```



4.2.4 Numerical Methods




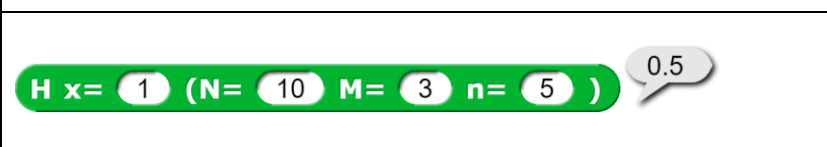

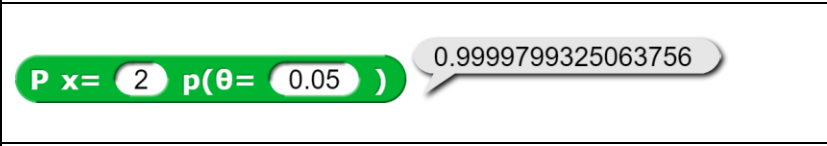

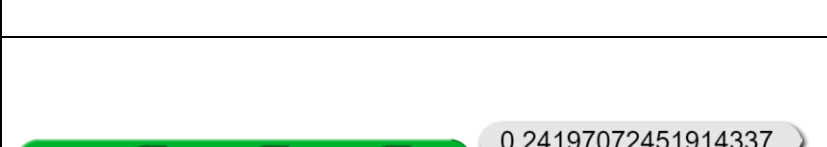
The library contains some blocks for dealing with sequences, series, secants, integrals and roots, as well as the calculation of a derivative at a given point.

	<p>Root calculation according to Newton's method. The term must be entered with a gray ring ("ringified"). <u>Example:</u> Calculation of the root of $f(x) = x^3 - 3x$, start at $x=1$.</p>
	<p>Calculation of a sequence element. The term must be entered with a gray ring ("ringified"). <u>Example:</u> 17th element of the sequence $\frac{1}{\sqrt{n}}$.</p>
	<p>Returns the first n elements of a sequence as a list. <u>Example:</u> The first 10 elements of the sequence $\frac{1}{\sqrt{n}}$.</p>
	<p>Sequence of secant slopes in a point. The sequence can also be specified explicitly in the form of a list. The term must be entered with a gray ring ("ringified"). <u>Example:</u> 10 secant slopes near the point with $x=2$ for $f(x) = x^3 - 3x$, calculated with the sequence $\frac{1}{n^2}$.</p>
	<p>Numerical calculation of the derivative in a point. The term must be entered with a gray ring ("ringified"). <u>Example:</u> Calculation of the derivative of $f(x) = x^3 - 3x$ at $x=0$.</p>
	<p>Calculates the sum of a finite series. The term must be entered with a gray ring ("ringified"). <u>Example:</u> The sum of the first 1000 elements of the series $\sum_{i=1}^{1000} \frac{1}{i^2}$.</p>
	<p>Numerical calculation of an integral using the trapezoidal method. The term must be entered with a gray ring ("ringified"). <u>Example:</u> Calculation of the integral $F = \int_0^\pi \cos x \, dx$ (with conversion to degrees)</p>

	Calculation of fast Fourier transformation (FFT) as well as their inverses (iFFT). Alternatively, the data of the frequency spectrum are calculated.
---	--

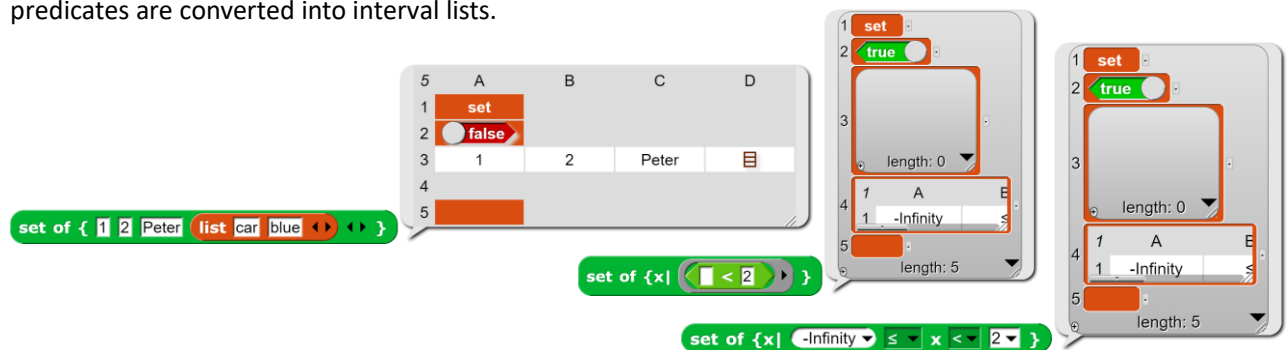
4.2.5 Statistics

For statistical applications *SciSnap!* contains a number of distributions. Correlation calculation, variances etc. are implemented in the data library, binomial coefficients and factorials can be found in the operators palette.

	Probabilities of the binomial distribution $b(N, p, k) = \binom{N}{k} \cdot p^k \cdot (1 - p)^{N-k}$
	Calculates the cumulative distribution function of the binomial distribution.
	Probabilities of the hypergeometric distribution $h(N, M, n, k) = \frac{\binom{M}{k} \cdot \binom{N-M}{n-k}}{\binom{N}{n}}$
	Calculates the cumulative distribution function of the hypergeometric distribution.
	Probabilities of the Poisson distribution $p(\theta, k) = \frac{\theta^k \cdot e^{-\theta}}{k!}$
	Calculates the cumulative distribution function of the Poisson distribution.
	Probabilities of the Pareto distribution $pareto(x_{min}, k, x) = \frac{k \cdot x_{min}^k}{x^{k+1}}; \\ x \geq x_{min}; 0 \text{ sonst}$
	Probabilities of the normal distribution $n(x, \mu, \sigma) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}};$

4.2.6 Sets

Sets are implemented in *SciSnap!* as 5-element lists. In the implementation, my goal was to deal with infinite sets as far as possible. For this, sets defined by predicates were somewhat neglected. To create a set, one can enumerate the elements $\{1, \text{car}, \text{true}\}$, define them via predicates $\{x/x < 5\}$ or enter ranges $\{-\text{Infinity} < x \leq 2\}$. The first element of a set contains the type ("set"), the second contains a predicate indicating whether it is a "numeric" set (i.e., all elements are numbers), the third is either empty or enumerates elements, the fourth may contain a list of intervals containing the set elements, and the last may contain a predicate. Compound predicates are again lists which contain the Boolean operator ("NOT", "OR", "AND") in the first element and then one or two predicates, which again may be compound. If possible, the list elements are described by intervals. For this purpose, e.g. predicates are converted into interval lists.



For further work with sets, the known set operations are available. If predicates are used, then only numbers and strings are useful as elements. If the sets can be described by intervals, then the operations are not limited. About 20 auxiliary blocks for e.g. interval operations do not appear in the palettes. You can find them when you select blocks to export.

	Blocks for defining sets (as described above).
	Returns "true" if the element is an element of the set, otherwise "false".
	The basic set operations for calculating the intersection, union, difference of sets, or the Cartesian product.
	Returns "true" if set1 is subset of set2, otherwise "false".
	Returns "true" if set1 = set2, otherwise "false".
	Represents the elements of a set "a bit more readable".
	Creates the corresponding list of elements from a text. Lists are bracketed "square" in the text, sets are bracketed "curly".

Some operations have to be performed with finite sets, e.g. to enumerate set elements. For this reason there is an upper bound for set elements, which is set by the *SciSnap!Properties*.

	Sets the limit up to which predicates or interval sizes are checked, if necessary.
	Returns the first n elements of a set as a list.

Examples:

Examples of SciSnap! blocks and their results:

- set set1 to set of { 1 5 27 30 44 }**
- set set2 to set of {x| < 6 }**
- set set3 to set of {x| -Infinity ≤ x ≤ 3 }**
- 10 elements of set1 ∩ set2 (numbers ∈ |N)**
 - Result: 27, 30, 44 (length: 3)
- 5 elements of set1 ∩ set3 (numbers ∈ |N)**
 - Result: 1, 5 (length: 2)
- element set1 [3] set of { 3 4 } ⇒ text**
 - Result: {[1,3],[1,4],[5,3],[5,4],[27,3],[27,4],[30,3],[30,4],[44,3],[44,4]}
- item 4 of set of {x| < 5 } ∩ set of { 3 }**
 - Result: Table with 4 columns (A, B, C, D) and 2 rows.

	A	B	C	D
2	-Infinity	≤	<	3
1	3	<	<	5


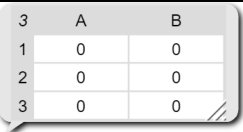

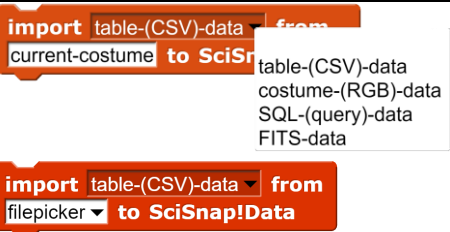

4.3 The Data Library

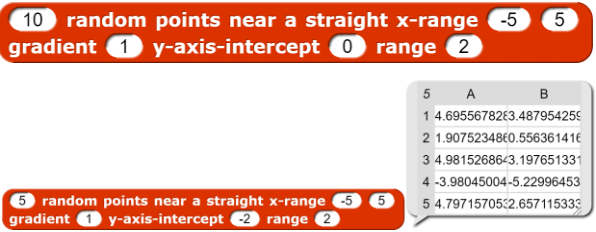
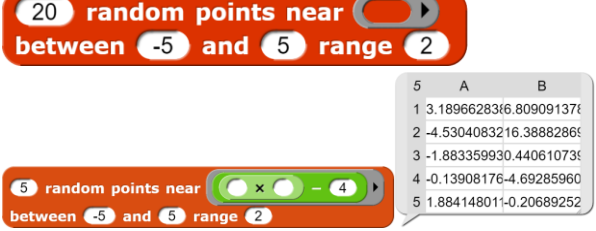

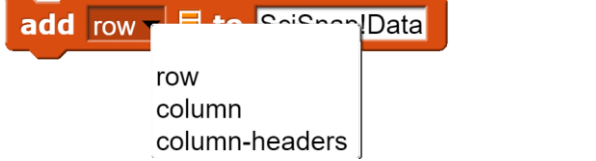
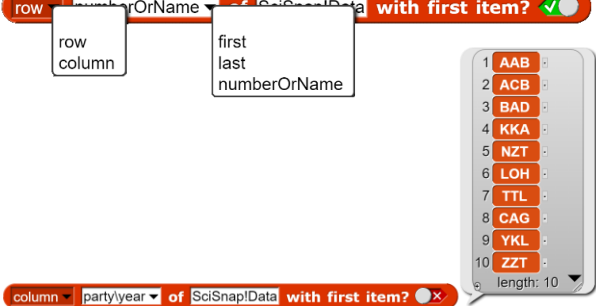
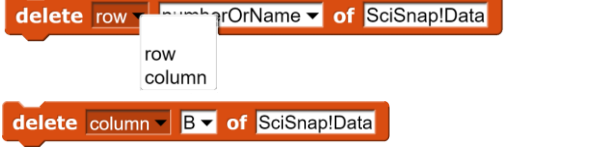
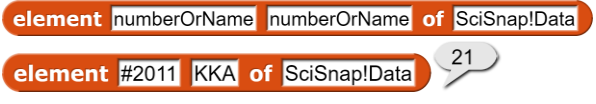

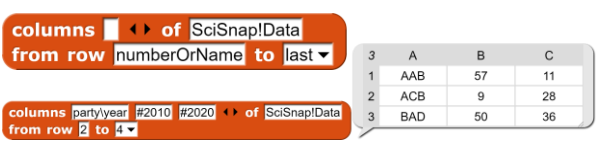
The data library of *SciSnap!* on the one hand serves for the direct manipulation of also larger data sets, on the other hand for the evaluation of data, e.g. for the computation of statistical quantities such as the covariance or correlations. In addition, there are some blocks for the standard machine learning methods.

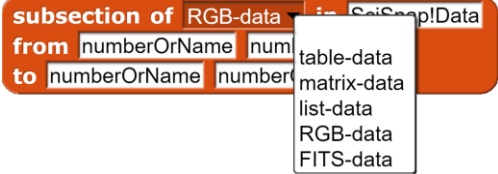
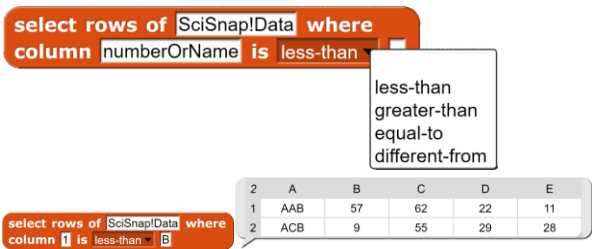



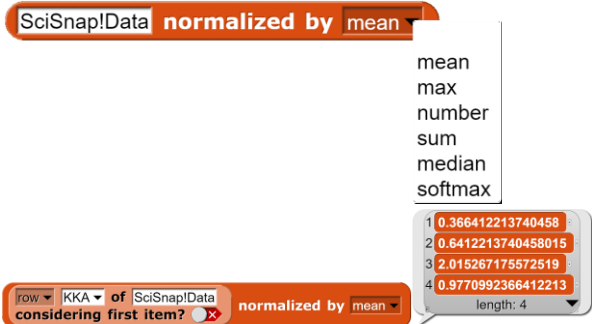

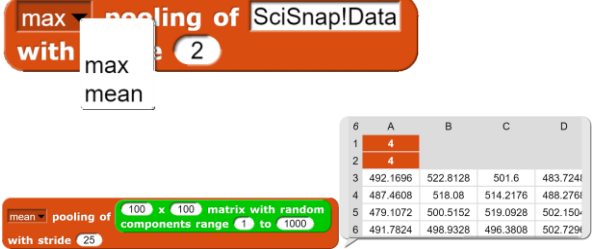

Because number structures such as vectors and matrices are implemented in the *math* library, the *data* library is largely limited to *tables* as an additional structure. Their rows and columns can be identified either by their numbers or the identifiers of the first column or row. Columns can additionally be named with capital letters (A..Z). If a number is required as identifier (i.e. not the column or row number), then a double cross (#) must be placed in front of the number as identifier, e.g. #123.

SciSnap!Data					
11	A	B	C	D	E
1	party/year	2010	2011	2014	2020
2	AAB	57	62	22	11
3	ACB	9	55	29	28
4	BAD	50	33	10	36
5	KKA	12	21	66	32
6	NZT	45	25	48	49
7	LOH	53	27	43	50
8	TTL	61	36	46	24
9	CAG	18	34	45	61
10	YKL	5	60	41	18
11	ZZT	26	12	39	56




The following examples refer to a table that contains "party names" as the first column, followed by "election results" in the years indicated.

 <p>empty table</p>	Returns an empty table (as a start structure for further table operations).
 <p>2 x 3 table initialized with 0</p>	Returns a table of the specified size, initialized with a starting value.
 <p>new 2 by 0 table with labels:</p> <p>new 2 by 3 table with labels: name age</p>	Returns a table of the specified size without contents, but with column headers.
<p>copy of</p>	Returns a copy of a list or table. Since <i>Snap!</i> assigns lists as references, you can use this block to avoid unintended changes to the original.
 <p>import table-(CSV)-data from current-costume to SciSnap!Data</p> <p>import table-(CSV)-data from filepicker to SciSnap!Data</p>	The block imports tables, image data or SQL data into the global variable <i>SciSnap!Data</i> , with which the other blocks of the library work by default. <u>Example:</u> Importing a table using the file selection dialog.
<p>write SciSnap!Data to CSV-file filename</p>	Writes a table to a file of the browser's download area under the specified name.
 <p>100 random points with ranges x: -100 100 y: -100 100 inside of a square</p> <p>square circle ring</p>	Returns n randomly distributed points of the specified range.

	<p>Returns n points scattering around a straight line, given by slope and y-axis intercept, in a range bounded by "range". Mostly used for tests.</p> <p><u>Example:</u> 5 points scattering around $f(x) = x - 2$.</p>
	<p>Returns n points that scatter around an arbitrary function, given by its "ringified" operators, in a range bounded by "range". Mostly used for tests.</p> <p><u>Beispiel:</u> 5 points scattering around $f(x) = x^2 - 4$.</p>
	<p>Returns a transposed table or list as result, i.e. one in which rows and columns have been interchanged.</p>
	<p>Adds a row, a column or column headers to the specified table. Missing elements are added with empty content, "overhanging" are ignored.</p>
	<p>Returns a row or column of the specified table.</p> <p><u>Example:</u> The first column of the example table without the header.</p>
	<p>Deletes a row or column from the specified table.</p> <p><u>Example:</u> Deletes the column for the year 2010 from the example table.</p>
	<p>Returns the specified table element.</p> <p><u>Example:</u> Result of the party KKA in 2011.</p>
	<p>Sets the value in a table cell.</p>
	<p>Returns the specified range of rows.</p> <p><u>Example:</u> Results of the three specified parties from 2010 and 2020.</p>

	<p>Returns a section of a table, matrix, list or image data given by the two points "left-top" and "right-bottom".</p>
	<p>Returns selected rows of a table that satisfy the specified criterion.</p> <p><u>Example:</u> All election results with parties starting with "A".</p>
	<p>Counts the occurrence of the values of a list.</p>
	<p>Determines the entropy of a list.</p>
	<p>Removes duplicates from a list.</p>
	<p>Normalize a vector by dividing it by the mean, maximum, number, sum, median of its values or by applying the softmax function.</p> <p><u>Example:</u> Results of the party KKA, "normalized" by the mean value.</p>
	<p>Compresses a vector or matrix with the specified factor by averaging.</p>
	<p>Returns a table that has been compressed by max- or mean-pooling with stride n. The dimensions of the new table are passed before the result. Well applicable to image data.</p> <p><u>Example:</u> A matrix of 100x100 random numbers is compressed with stride 25.</p>
	<p>Sorts a list using the specified predicate.</p>








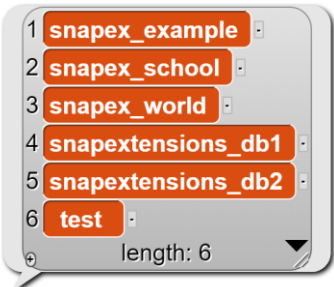





<div>SciSnap!Data sorted by column numberOrName ascending considering headline?</div> <div><table><tr><td>12</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td></tr><tr><td>1</td><td>party/year</td><td>2010</td><td>2011</td><td>2014</td><td>2020</td></tr><tr><td>2</td><td>AAB</td><td>57</td><td>62</td><td>22</td><td>11</td></tr><tr><td>3</td><td>AAB</td><td>57</td><td>62</td><td>22</td><td>11</td></tr><tr><td>4</td><td>YKL</td><td>5</td><td>60</td><td>41</td><td>18</td></tr><tr><td>5</td><td>ACB</td><td>9</td><td>55</td><td>29</td><td>28</td></tr><tr><td>6</td><td>TTL</td><td>61</td><td>36</td><td>46</td><td>24</td></tr><tr><td>7</td><td>CAG</td><td>18</td><td>34</td><td>45</td><td>61</td></tr><tr><td>8</td><td>BAD</td><td>50</td><td>33</td><td>10</td><td>36</td></tr><tr><td>9</td><td>LOH</td><td>53</td><td>27</td><td>43</td><td>50</td></tr><tr><td>10</td><td>NZT</td><td>45</td><td>25</td><td>48</td><td>49</td></tr><tr><td>11</td><td>KKA</td><td>12</td><td>21</td><td>66</td><td>32</td></tr><tr><td>12</td><td>ZZT</td><td>26</td><td>12</td><td>39</td><td>56</td></tr></table></div> <div>SciSnap!Data sorted by column #2011 ascending considering headline?</div>	12	A	B	C	D	E	1	party/year	2010	2011	2014	2020	2	AAB	57	62	22	11	3	AAB	57	62	22	11	4	YKL	5	60	41	18	5	ACB	9	55	29	28	6	TTL	61	36	46	24	7	CAG	18	34	45	61	8	BAD	50	33	10	36	9	LOH	53	27	43	50	10	NZT	45	25	48	49	11	KKA	12	21	66	32	12	ZZT	26	12	39	56	<div>Returns a table sorted in ascending or descending order by the specified column.</div> <div>Example: The election results sorted by the year 2011.</div>
12	A	B	C	D	E																																																																										
1	party/year	2010	2011	2014	2020																																																																										
2	AAB	57	62	22	11																																																																										
3	AAB	57	62	22	11																																																																										
4	YKL	5	60	41	18																																																																										
5	ACB	9	55	29	28																																																																										
6	TTL	61	36	46	24																																																																										
7	CAG	18	34	45	61																																																																										
8	BAD	50	33	10	36																																																																										
9	LOH	53	27	43	50																																																																										
10	NZT	45	25	48	49																																																																										
11	KKA	12	21	66	32																																																																										
12	ZZT	26	12	39	56																																																																										
<div>mean of column numberOrName of SciSnap!Data grouped by column numberOrName considering headline?</div> <div><div>min max number sum mean</div><div><table><tr><td>10</td><td>A</td><td>B</td></tr><tr><td>1</td><td>value</td><td>mean</td></tr><tr><td>2</td><td>AAB</td><td>57</td></tr><tr><td>3</td><td>ACB</td><td>9</td></tr><tr><td>4</td><td>BAD</td><td>50</td></tr><tr><td>5</td><td>CAG</td><td>18</td></tr><tr><td>6</td><td>KKA</td><td>12</td></tr><tr><td>7</td><td>LOH</td><td>53</td></tr><tr><td>8</td><td>NZT</td><td>45</td></tr><tr><td>9</td><td>TTL</td><td>61</td></tr><tr><td>10</td><td>YKL</td><td>5</td></tr></table></div><div>mean of column #2010 of SciSnap!Data grouped by column 1 considering headline?</div><div><table><tr><td>4</td><td>A</td><td>B</td></tr><tr><td>1</td><td>value</td><td>mean</td></tr><tr><td>2</td><td>2010</td><td>57</td></tr><tr><td>3</td><td>2011</td><td>62</td></tr><tr><td>4</td><td>2014</td><td>22</td></tr></table></div><div>mean of column AAB of transpose table or list SciSnap!Data grouped by column 1 considering headline?</div></div>	10	A	B	1	value	mean	2	AAB	57	3	ACB	9	4	BAD	50	5	CAG	18	6	KKA	12	7	LOH	53	8	NZT	45	9	TTL	61	10	YKL	5	4	A	B	1	value	mean	2	2010	57	3	2011	62	4	2014	22	<div>Returns minimum, maximum, number, sum or average of the first specified column of data, grouped by the second. The headings can be included - or not.</div> <div>Examples: The mean values for 2010, grouped by party. The mean values for 2010, grouped by party.</div>																														
10	A	B																																																																													
1	value	mean																																																																													
2	AAB	57																																																																													
3	ACB	9																																																																													
4	BAD	50																																																																													
5	CAG	18																																																																													
6	KKA	12																																																																													
7	LOH	53																																																																													
8	NZT	45																																																																													
9	TTL	61																																																																													
10	YKL	5																																																																													
4	A	B																																																																													
1	value	mean																																																																													
2	2010	57																																																																													
3	2011	62																																																																													
4	2014	22																																																																													
<div>ranges of column numberOrName and numberOrName of SciSnap!Data considering headline?</div> <div><div>ranges covariance correlation</div><div>correlation of column #2010 and #2011 of SciSnap!Data considering headline?<div>0.047376831823748154</div></div></div>	<div>Returns ranges, covariance and correlation between two table columns.</div> <div>Example: Correlation between the years 2010 and 2011.</div>																																																																														
<div>regression line parameters of SciSnap!Data</div> <div>regression line parameters of 10 random points near a straight line x-range -5 5 gradient 1 y-axis-intercept 0 range 2<div><div>1 0.9902951104347184 2 0.31005583661222125</div><div>length: 2</div></div></div>	<div>Returns gradient and y-axis intercept of the regression line through the specified data.</div> <div>Example: Regression line through 10 random points scattering around a straight line.</div>																																																																														
<div>5 next neighbors of in SciSnap!Data</div>	<div>k-nearest neighbor (kNN) method in two dimensions for machine learning.</div> <div>Example: HR-diagram</div>																																																																														
<div>convolution kernel applied to table SciSnap!Data width 100 height 100</div> <div>image table</div>	<div>Returns the result of a convolution, applied either to a table or to image data.</div> <div>Examples: edge detection, CNN</div>																																																																														
<div>3 -means clustering for SciSnap!Data with Euclidean metrics</div> <div><table><tr><td>10</td><td>A</td><td>B</td><td>C</td></tr><tr><td>1</td><td>-19</td><td>-8</td><td>1</td></tr><tr><td>2</td><td>64</td><td>-41</td><td>3</td></tr><tr><td>3</td><td>74</td><td>-27</td><td>3</td></tr><tr><td>4</td><td>9</td><td>30</td><td>1</td></tr><tr><td>5</td><td>-13</td><td>-37</td><td>1</td></tr><tr><td>6</td><td>-100</td><td>60</td><td>2</td></tr><tr><td>7</td><td>-94</td><td>-13</td><td>2</td></tr><tr><td>8</td><td>-93</td><td>70</td><td>2</td></tr><tr><td>9</td><td>-38</td><td>13</td><td>1</td></tr><tr><td>10</td><td>69</td><td>67</td><td>3</td></tr></table></div> <div>3 -means clustering for 10 random points with ranges for x: -100 100 y: -100 100 with Euclidean metrics</div>	10	A	B	C	1	-19	-8	1	2	64	-41	3	3	74	-27	3	4	9	30	1	5	-13	-37	1	6	-100	60	2	7	-94	-13	2	8	-93	70	2	9	-38	13	1	10	69	67	3	<div>Clustering of n-dimensional data with the k-means method. The Euclidean distance is taken as the metric. Cluster numbers are appended to the data.</div>																																		
10	A	B	C																																																																												
1	-19	-8	1																																																																												
2	64	-41	3																																																																												
3	74	-27	3																																																																												
4	9	30	1																																																																												
5	-13	-37	1																																																																												
6	-100	60	2																																																																												
7	-94	-13	2																																																																												
8	-93	70	2																																																																												
9	-38	13	1																																																																												
10	69	67	3																																																																												
<div>3 -means clustering for SciSnap!Data with metric</div>	<div>Clustering with any metric.</div> <div>Example: DNS-Clustering with Levenshtein metric.</div>																																																																														


























Levenshtein-distance of Saturday and Sunday	Returns the Levenshtein distance between two strings.
DBSCAN clustering for SciSnap!Data radius 50 minMembers 5	Clusters data according to the density based DBSCAN method.
decision tree ID3 for  with labeled data in last column	Returns a decision tree for the specified data, constructed using the ID3 method.
classify  with ID3-tree 	Classifies data using an ID3 tree.

4.4 The SQL Library

The *SQL* library contains most of the commands required for *SQL* queries (*select*). Other *SQL* statements can be entered directly into the *exec SQL-command* block. However, in this case you must have the appropriate access rights. The library works with a global variable *SQLData*, which prevent the data stored in them from conflicting with that of other sprites. The variable is created automatically during configuration.

Similar to the *Math* and *Data* blocks, the *SQL* blocks do not work with a specific sprite or stage. However, each call first checks whether *SciSnap!* has been successfully configured for *SQL* access. If this is not the case, an error message is displayed - for reporter blocks as the result of the function call, for command blocks as the output of the calling sprite as well as in the *SciSnap!* collection box for error messages *SciSnap!Messages*.

	<p>Configures <i>SciSnap!</i> for <i>SQL</i> access. For this, the global variable <i>SQLData</i> is created and the initial properties are set. The sprite that executes the command will take the costume <i>SQLDisconnected</i> if it exists.</p> 
	<p>Returns "true" if the configuration is correct, otherwise "false".</p>
	<p>Connects to a database server whose address is in the script. This should be reset, e.g. as <i>localhost</i>, if the default server is not used. If the connection attempt is successful, the executing sprite will take the costume <i>SQLConnected</i> if it is present.</p> 
	<p>Imports a table into the <i>SQLData</i> data space.</p> <p><u>Example:</u> Import of a query result.</p>
<p></p>  <p></p>	<p>Returns a list of currently available databases.</p>
	<p>Selects one of the existing databases.</p>
<p></p>  <p></p>	<p>Returns a list of the tables of the selected database.</p>



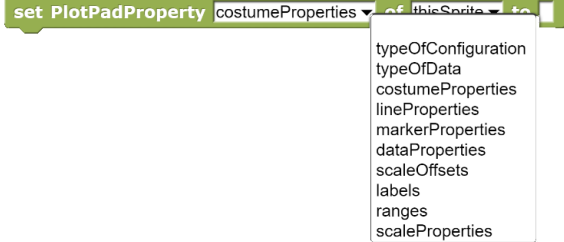
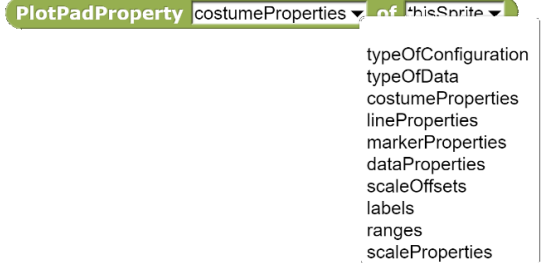

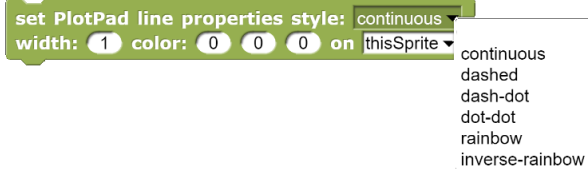
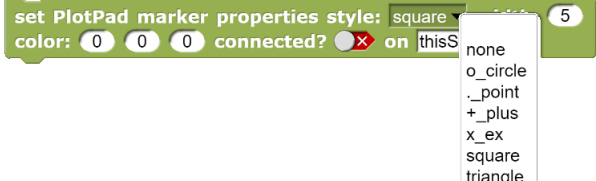



	Selects one of the existing tables.
  	Returns a list of attributes of the specified table.
   	Block for generating simple SQL queries that can be executed by the <i>exec SQL-command</i> block.
      	Block for generating complex SQL queries that can be executed by the <i>exec SQL-command</i> block.
	Block for executing SQL statements for a database. The statements can be generated by <i>select</i> blocks or entered directly.
	Predicates for the execution of comparisons.
	Predicates for the execution of logical operations.
	Predicate for checking a string pattern.
	Predicate to check if an element is included in an enumeration.
    	Aggregate functions.

The following *SciSnap!* libraries work with locally configured *special sprites*. These are conceptually derived from *sketchpads*, so they are used for making sketches, experimenting, trying out the effect of commands, etc. If the pad is too full, then a new "page" is taken from it and work continues.

Each sprite and the stage can serve as a *special sprite*. For this purpose they are configured accordingly by creating two local variables *myData* and *myProperties* and filling them with initial values. The commands that refer to a *special sprite* all contain the target of the operation - i.e. a sprite or the stage. Before executing the instructions, it is checked in each case whether the target has been configured correctly. After that, the local data and properties of the target are used. On the one hand, this procedure largely eliminates object-oriented calls, which greatly lengthen the instruction blocks if data must be passed, and on the other hand, it keeps the data local to the sprites that the operations concern. For example, if data is measured in an image and plotted in a graph, the *ImagePad* and the *PlotPad* can work independently with their own data and properties. The initial settings make it possible to work with halfway reasonable default settings. If the result shows that other settings would make more sense, then the default settings are changed - but only then. This way of working makes it possible to get by with relatively few parameters for the individual blocks. The properties are largely grouped together, for example the properties of the lines to be drawn. This means that they can be transferred or read "in one go" and their number is kept within limits.

4.5 The PlotPad Library

PlotPads are used to display graphs, histograms, etc. The current *SciSnap!PlotPad Library* was heavily designed by *Rick Hessman*, especially the *PrettyPrinting* and the *line styles* are from him. Many thanks for that!

	<p>Configures a sprite or the stage as a <i>PlotPad</i>. The name of "<i>anotherSprite</i>" must be specified if required. The command must be executed once before working with a sprite as a <i>PlotPad</i>. The target of the call takes a rectangular costume with the specified dimensions and colors.</p>
	<p>Returns "true" if the configuration is correct, otherwise "false".</p>
	<p>Sets one of the properties in <i>myProperties</i> to the specified value.</p>
	<p>Returns one of the properties in <i>myProperties</i>.</p>
	<p>Sets the costume properties of the specified target.</p>
	<p>Sets the line properties of the specified target.</p>
	<p>Sets the marker (datapoint) properties of the specified target.</p>
	<p>Sets the scale properties of the specified target.</p>
	<p>Sets the label properties of the specified target.</p>
	<p>Sets the distances of the coordinate axes from the edges.</p>

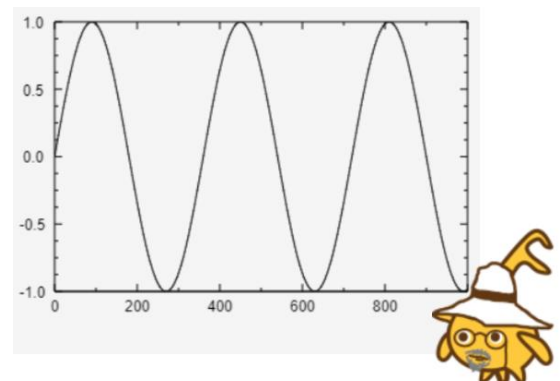
set PlotPad ranges for x: -10 10 y: -10 10 with border? <input checked="" type="checkbox"/> of 0.1 pretty formatted? <input checked="" type="checkbox"/> on thisSprite	Sets the ranges of the axes of the coordinate system.
add graph ringified-operator-or-polynomial to PlotPad thisSprite	Adds a function graph to the <i>PlotPad</i> , given as a list of polynomial coefficients or as a "ringified" term.
add dataplot of numeric data: myData to PlotPad thisSprite	Adds a data plot for a two-dimensional data table to the <i>PlotPad</i> .
add dataplot of mixed data: myData y-scale? <input checked="" type="checkbox"/> x-scale? <input checked="" type="checkbox"/> to PlotPad thisSprite	Adds a data plot to the <i>PlotPad</i> for a two-dimensional table containing texts in the first column and numerical values in the second.
add histogram of myData with 10 groups pretty formatted? <input checked="" type="checkbox"/> to PlotPad thisSprite	Adds a histogram to the <i>PlotPad</i> .
add axes and scales to PlotPad thisSprite	Adds axes and scales to the <i>PlotPad</i> . Depending on your preference, you can also add this block to the actual plot commands at the end.
clear plot of thisSprite	Deletes the <i>PlotPad</i> without changing the other pre-sets.
set pretty ranges on PlotPad thisSprite	Sets the ranges of the axes so that "pretty" labels are on the axes.
pretty values for a PlotPad from -10 to 10 with 6 intervals	Returns values for "pretty" labels of the axes.
get ranges for PlotPad thisSprite from myData with border 0.1	Redetermines the ranges, calculated from the data.
ranges of 2-dim table	Returns the ranges of a two-dimensional table.
convert value 100 to coordinate xp of PlotPad thisSprite	Returns the conversion of a numerical value into coordinates of the <i>plotpad</i> or the coordinate system used.
PlotPad costume-coordinates mouse costume-coordinates graph-coordinates	Converts the mouse position into costume or graph coordinates.
Example 3: Simple plot of data: x: 0 y: 0 width: 600 height: 400 title: labels: line: continuous marker: square color: 0 0 0	Simple plot program for data.

Example:

```

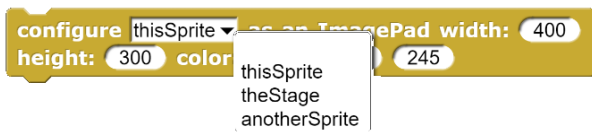




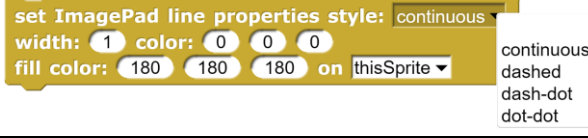

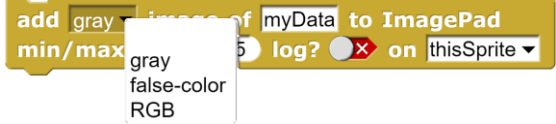
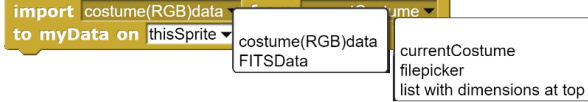
configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad ranges for x: 0 1000 y: -1 1
with border? ☒ of 0.1 pretty formatted? ☒
on thisSprite
add graph sin of 0 to PlotPad thisSprite
add axes and scales to PlotPad thisSprite

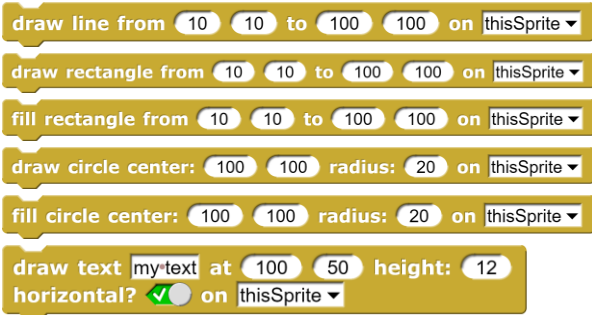

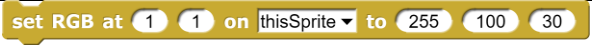



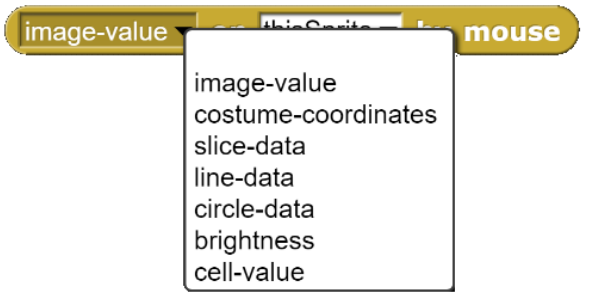





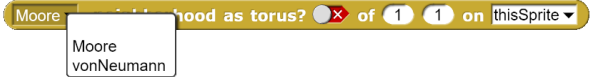

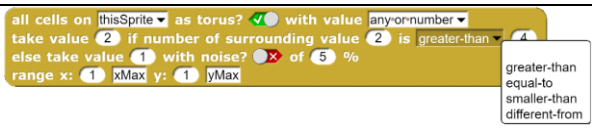
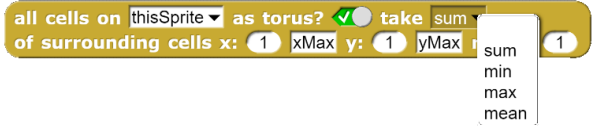
```

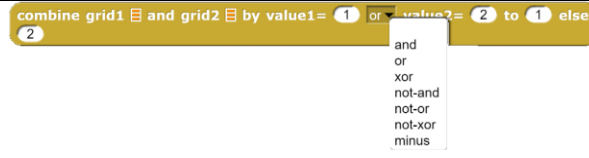



4.6 The ImagePad Library

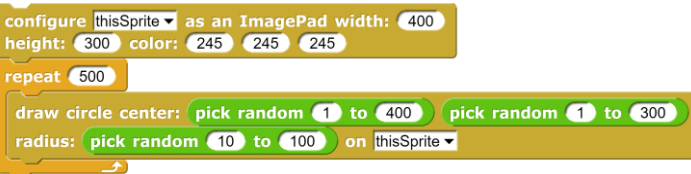
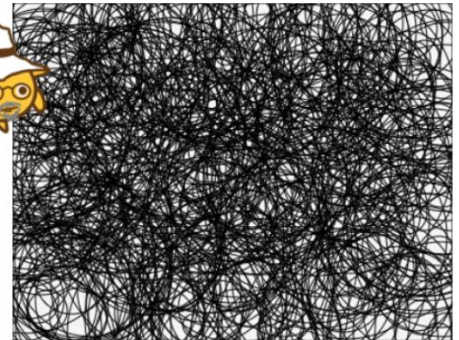
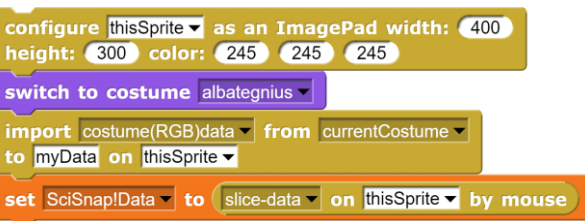
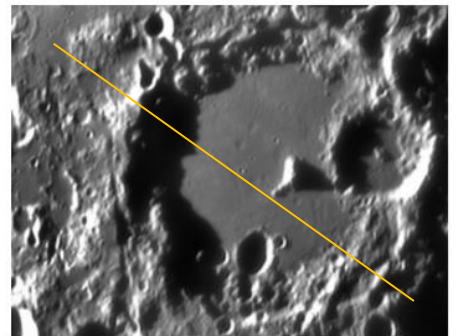
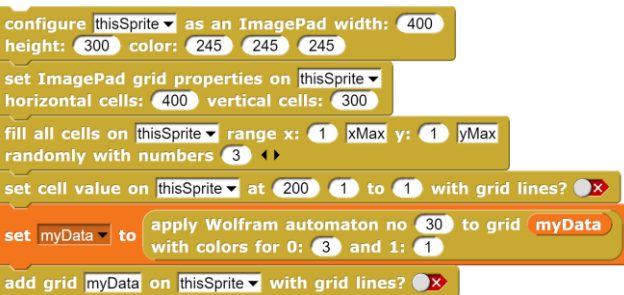
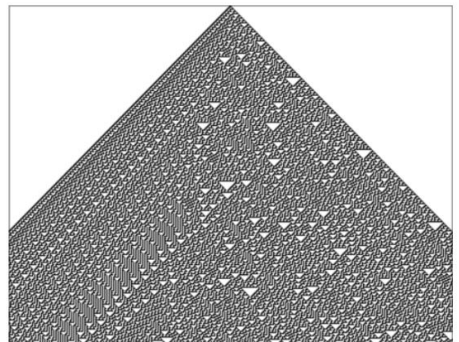
ImagePads are used to display image data, i.e. to generate images. Measurements can be made in these images using the mouse - for example, a section can be created through the image. In addition, some of the usual operations for drawing lines, rectangles, circles, and texts are available. In addition, grids can be displayed, and operations can be performed on grids. The coordinate system on *ImagePads* is the usual one for images: the origin is in the upper left corner and the y-axis is directed downwards.

	<p>Configures a sprite or the stage as <i>ImagePad</i>. The name of "anotherSprite" must be specified if required. The command must be executed once before working with a sprite as <i>ImagePad</i>. The target of the call takes a rectangular costume with the specified dimensions and colors.</p>
	<p>Returns "true" if the configuration is correct, otherwise "false".</p>
	<p>Returns one of the properties in <i>myProperties</i>.</p>
	<p>Sets one of the properties in <i>myProperties</i> to the specified value.</p>
	<p>Sets the costume properties of the specified target.</p>
	<p>Sets the line properties of the specified target.</p>
	<p>Sets the values for the dimensions of the grid.</p>
	<p>Creates an image from the image data on a <i>ImagePad</i>.</p>
	<p>Imports image data to <i>myData</i> of the specified sprite.</p>

	Elementary drawing operations.
	Draws a list of "points" as circles or squares. Attention: JS-coordinates are used!
	Sets a pixel of the costume to the specified value.
	Returns the value of a pixel of the costume.
	Sets an image point in the data space to the specified value.
	Returns the value of an image point from the data space.
	Returns image data using the mouse: individual image values, image coordinates, sections through the image, end points of a line or data of a circle, brightness values from a range or a grid value. The image data must be located in <i>myData</i> .
	Returns the result of an affine transformation of a costume described by specifying three original points and three image points.
	Returns the total brightness around a pixel in the specified radius.
	Randomly fills the cells of a grid with one of the specified numbers.
	Set cell contents using the mouse.
	Direct setting of cell contents.
	Determines the specified neighborhood of a grid cell.
	Randomly swaps the values of grid cells with those of neighboring cells from the specified range.
	Replaces the cell values from the specified range under the specified conditions.
	Replaces the cell values from the specified range with the result of the specified aggregate function.

	<p>Superposes two grids using the specified operation.</p>
	<p>Applies the operations of the given linear Wolfram automaton on a grid from the first line to all following ones.</p>

Examples:

4.6 Die GraphPad-Bibliothek

Graphs are one of the most powerful models in computer science. With their help, complex systems can be studied, especially the effect of the interconnection of numerous similar subsystems. However, the algorithms required for this, such as breadth-first search or routing, are themselves non-trivial, so it seems reasonable to provide them as basic commands in order to concentrate the work on the modeling itself. This is exactly the purpose of the *GraphPads*.

	Configures a sprite or the stage as a <i>GraphPad</i> . The name of " <i>anotherSprite</i> " must be specified if required. The command must be executed once before working with a sprite as <i>GraphPad</i> . The target of the call takes a rectangular costume with the specified dimensions and colors.
	Returns "true" if the configuration is correct, otherwise "false".
	Sets one of the properties in <i>myProperties</i> to the specified value.
	Returns one of the properties in <i>myProperties</i> .
	Sets the costume properties of the specified target.
	Sets the vertex (node) properties of the specified target.
	Sets the edge properties of the specified target.
	Adds n vertices (nodes) to the graph at random positions.
	Adds one vertex (node) to the graph at the specified position.
	Moves a vertex (node) to the specified position.
	Adds n random edges to the graph.
	Adds an edge to the graph between the specified vertices (nodes).
	Draws the graph. Colors connected vertices (nodes) in the same color. If you draw on the stage, the initial background image remains, e.g. to be able to use maps.
	Deletes a vertex (node) of the graph.
	Deletes an edge of the graph.

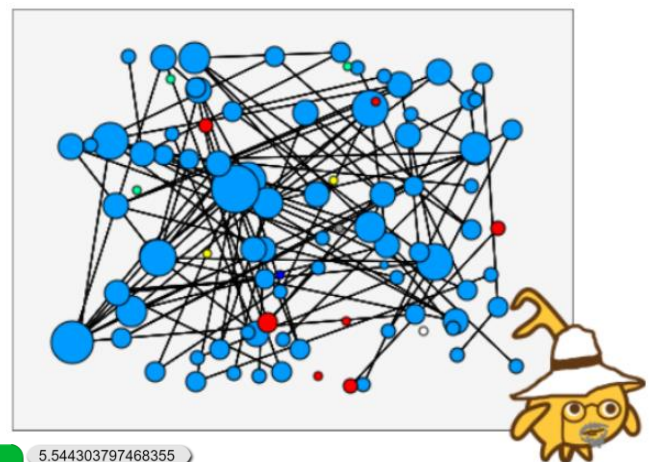
weight of edge from vertex 1 to vertex 2 of graph on thisSprite ▾	Returns the weight of an edge, if possible.
change weight of edge from vertex 1 to vertex 2 to 1 of graph on thisSprite ▾	Changes the weight of an edge, if possible.
ask for new weight of graph on thisSprite ▾	Asks for a new edge weight.
ask for new start vertex width of graph on thisSprite ▾	Asks for a new start vertex size.
content of vertex 1 of graph on thisSprite ▾	Returns the content of a vertex (node).
change content of vertex 1 to of graph on thisSprite ▾	Changes the content of a vertex (node).
ask for new vertex content in graph on thisSprite ▾	Asks for a new vertex (node) content.
set marker of vertex 1 of graph on thisSprite ▾	Marks a vertex (node).
remove marker of vertex 1 of graph on thisSprite ▾	Removes the marking of a vertex (node).
remove all markers of graph on thisSprite ▾	Removes all markers in the graph.
depth first search of content starting at vertex 1 of graph on thisSprite ▾	Depth-first search for a content starting from the vertex with the specified number.
breadth first search of content starting at vertex 1 of graph on thisSprite ▾	Breadth-first search for a content starting from the vertex with the specified number.
distance on thisSprite ▾ from vertex 1 to vertex 2	Spatial distance between two vertices on the sprite.
shortest path in graph from vertex 1 to vertex 2 on thisSprite ▾	Shortest path between two vertices in the graph.
list of all shortest paths in graph from vertex 1 to all connected vertices of graph on thisSprite ▾	List of all shortest paths from a vertex to all connected vertices in the graph.
vertexnumber at 100 50 of graph on thisSprite ▾	Number of a vertex at the given position on the sprite.
point 0 0 on sprite/stage → point on graph thisSprite ▾	Converts stage coordinates to graph (JS-style) coordinates.
vertexnumber of Peter in graph of thisSprite ▾	Returns the number of the vertex with the specified content.
vertexnumber of graph on thisSprite ▾ at mouse position thisSprite theStage anotherSprite	Returns the number of the node at the mouse position.

Examples:

```

configure thisSprite ▾ as a GraphPad width: 400
height: 300 color: 245 245 245
add 100 random vertices to graph on thisSprite ▾
add 100 random edges to graph on thisSprite ▾

```



```

mean ▾ of vector
column ▾ 3 ▾ of list of all shortest paths in graph from vertex 1
first item? ▾ ✓ of to all connected vertices of graph on thisSprite ▾ with

```

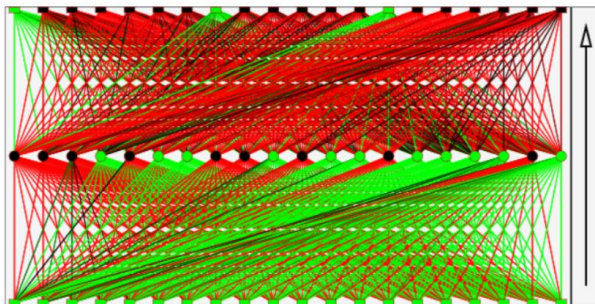

4.7 The NeuralNetPad Library

The basic principles of neural networks are simple and obvious, but the learning procedures of the systems with their discrete gradient descent and the associated partial derivatives are not so easy to understand for those who are distant from mathematics. In particular, it is not so easy to understand what a neural network has actually learned. The *SciSnap!NNPadLibrary* is intended to enable the handling of neural networks on an intermediate level between very small and really large networks. It illustrates the values of the edges by coloring, where positive values are shown in green and negative values in red colors. Small values tend to be black. The library facilitates the creation and training of fully connected perceptron nets.

	Configures a sprite or the stage as <i>NNPad</i> . The name of "anotherSprite" must be specified if required. The command must be executed once before working with a sprite as <i>NNPad</i> . The target of the call takes a rectangular costume with the specified dimensions and colors.
	Returns "true" if the configuration is correct, otherwise "false".
	Sets one of the properties in <i>myProperties</i> to the specified value.
	Returns one of the properties in <i>myProperties</i> .
	Sets the costume properties of the specified target.
	Sets the layer properties of the specified target.
	Adds random weights for an NN of the specified width and depth.
	Returns the output of the nth layer of the NN for the specified input vector.
	Displays the status of the network for the specified input vector in color-coded form.
	Trains the NN for the given input vector to achieve the given output vector.

Example: A neural network is trained to produce the specified pattern when the numbers 1 to 20 are applied to the inputs.

```
configure thisSprite as a NeuralNetPad width: 600  
height: 300 color: 245 245 245  
NN add new weights for 2 layers of width 20 on thisSprite  
repeat 200  
  teach NN with input numbers from 1 to 20 and target output  
  list 1 0 0 -1 0 0 0 1 0 0 0 0 0 1 -1 -1 0 0 0 by back-  
  propagation with learning factor 0.1 on thisSprite  
NN show status with input numbers from 1 to 20 on thisSprite
```

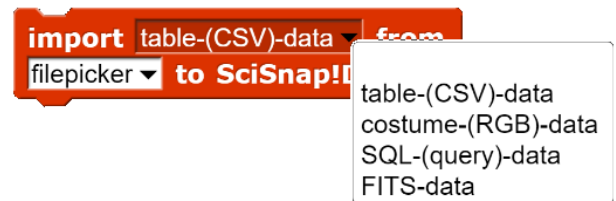


5 Data Import and Export

Snap! can import a number of data formats directly. This can be done by "dropping" appropriate files onto the *Snap!* window or importing them by right-clicking on a Variable-Watcher¹⁶. Both work well with *text*, *CSV* and *JSON* files. Other text file formats like *FITS* can also be imported this way, asking if you are serious. Exporting works the same way. The data will then end up in the download folder of the browser. If you want to do the same programmatically, use the option *read file with filepicker*. A file manager window appears where you select the file as usual. After that the data will be imported.

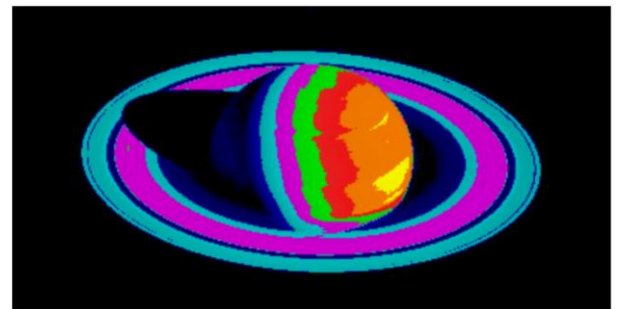
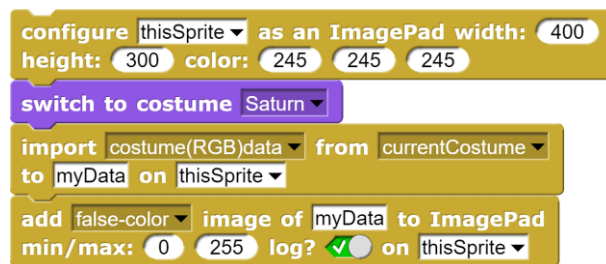


The essential task remains to assign this data to the *SciSnap!Data* variable and set the corresponding properties in *SciSnap!Properties*. This is done by the following block, which imports data from outside into the *SciSnap!Data* area. This can be image data, table data or the data of the current costume. This is stored as a table of RGB values.



Example: False color image

Using an *ImagePad*, an image (source: [NASA]) is saved and redisplayed with false colors.



Example: CSV Import

Almost 600000 records from a CSV file are read in about 10 seconds. The properties are set.



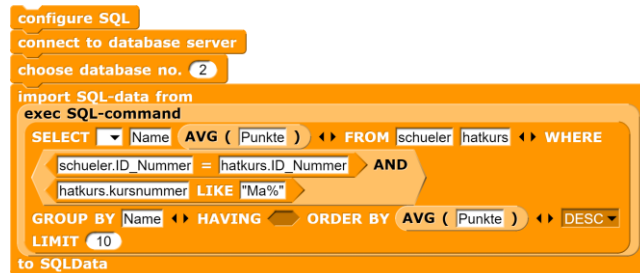
SciSnap!Properties		
1	typeOfData	table
2	width	not set
3	height	not set
4	minValue	not set
5	maxValue	not set
6	columns	15
7	rows	577704
8	maxSetValue	500

SciSnap!Data			
577704	A	B	C
1	tripduration	starttime	stoptime
2	695	2013-06-01	(2013-06-01)
3	693	2013-06-01	(2013-06-01)
4	2059	2013-06-01	(2013-06-01)
5	123	2013-06-01	(2013-06-01)
6	1521	2013-06-01	(2013-06-01)
7	2028	2013-06-01	(2013-06-01)
8	2057	2013-06-01	(2013-06-01)

¹⁶ A variable watcher is obtained by checking the box next to the variable.

Example: SQL Import

If we have access to an SQL server, we can also import data from it. In our case, we use the *SciSnap!* SQL library to import the results of a query into the *SQLData* variable. In doing so, the data is converted into table form and its relevant properties such as number of columns and rows, ... are reset in the *SciSnap!Properties*.

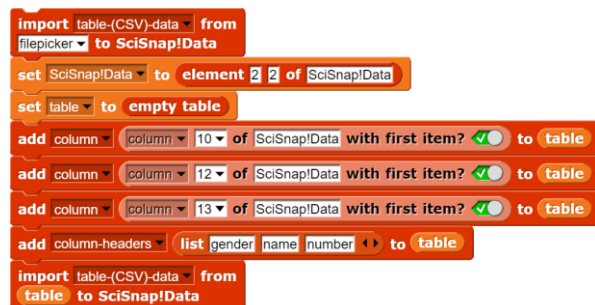


16	A	B
1	typeOfData	table
2	width	not set
3	height	not set
4	minValue	notSet
5	maxValue	notSet
6	columns	2
7	rows	10
8	maxSetValue	500
9	typeOfConfig	SQL
10	connection	https://snape
11	connected	<input checked="" type="checkbox"/>
12	databases	
13	currentDatabase	snape_sch
14	tables	
15	currentTable	
16	attributes	

10	A	B
1	Kirsche	13.7500
2	Pogenberg	13.5000
3	Rassin	13.5000
4	Karbel	12.7500
5	Krahn	12.0000
6	Rawe	12.0000
7	Gallus	11.7500
8	Ruf	11.5000
9	Boemmel	11.5000
10	Vestmann	11.0000

Example: JSON Import

Again, the easiest way is to simply "drop" a *JSON* file into the *Snap!* window. But it can also be done automatically. First of all, we look for interesting JSON data and of course choose the *statistics of baby names in New York City* for this - what else. The appropriate block for this is again *import <table data> from <filepicker> to SciSnap!Data*. The result is a list with two columns and two rows, the metadata and the actual data. Because we are interested in these we replace the original data with the element (2|2) of the table. Of course, we looked at the individual elements in table form beforehand to check what we loaded there in the first place. From the many columns we copy the three interesting ones into a new table, add column headers and import the result back into *SciSnap!Data*.



The result:

19419 baby names - Who would have thought it!

2	A	B
1	meta	
2	data	

19419	A	B	C
1	gender	name	number
2	FEMALE	Olivia	172
3	FEMALE	Chloe	112
4	FEMALE	Sophia	104
5	FEMALE	Emily	99
6	FEMALE	Emma	99
7	FEMALE	Mia	79
8	FEMALE	Charlotte	59
9	FEMALE	Sarah	57
10	FEMALE	Isabella	56
11	FEMALE	Hannah	56
12	FEMALE	Grace	54
13	FEMALE	Angela	54
14	FEMALE	Ava	53
15	FEMALE	Joanna	49

Examples: Data import by mouse

In many cases, especially with images, it is advantageous to read in data using the mouse. For this purpose, *ImagePads* have a block that can be used to determine image values, image coordinates, the data on a section through the image, the start and end points of a line, the center and radius of a circle, and the summed brightness values together with their number in a circle. As an example, the height of ancient columns is to be measured. For this purpose, the costume image of the *ImagePad* with the columns is imported and then measured with the mouse (yellow line).

data		
	A	B
2		
1	122	42
2	125	172

```

configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
switch to costume columns
import costume(RGB)data from currentCostume
to myData on thisSprite
set data to line-data on thisSprite by mouse

```



As a second example for measuring with the mouse we want to measure the total brightness inside a circle around a star photo (source: [HOU]).

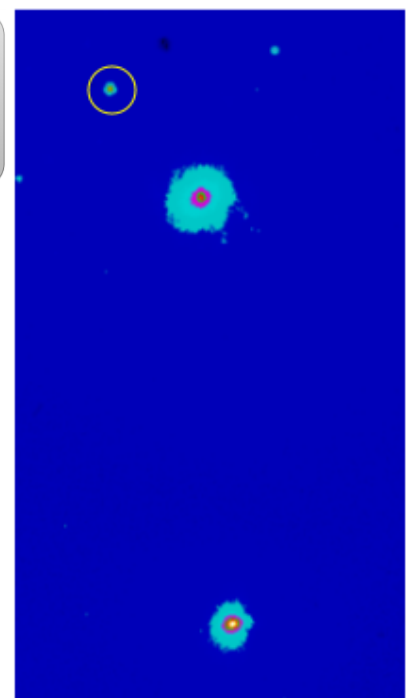
```

configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
import FITSData from filepicker
to myData on thisSprite
add false-color image of myData to ImagePad
min/max: 0 700 log? on thisSprite
set data to brightness on thisSprite by mouse

```

For grayscale images like FITS we get the total brightness and the number of measured pixels, for RGB images we get the brightness of the three colors and the number of pixels.

data	
1	952
2	15
length: 2	



The export of data again can be done directly from a variable watcher.

For scripts there are two new blocks *write <table> to CSV file <filename>* and *write string <string> to file <filename>*. The results end up in the browser's download folder, as usual in *Snap!* The two blocks allow to automate the data exchange with spreadsheet programs or text files, for example to save the results of data processing.

98	A	B
1	1.2	188.4
2	6	231.2
3	11	185.4
4	16	144.2
5	21	27.4

write SciSnap!Data **to CSV-file** filename

write text this-text **to TXT-file** this-file

6 Math related Examples

6.1 Representation of complex numbers

The operations with complex numbers can be easily illustrated in *SciSnap!* by using the *MathPad*: a sprite configuration that allows to quickly represent, for example, complex numbers as arrows. Since the complex plane has two dimensions, we need to change the default (3 dimensions), then we represent two complex numbers and their sum in different colors.

Create the *MathSprite* sprite as *MathPad* in the specified size and color. Then set dimension etc.

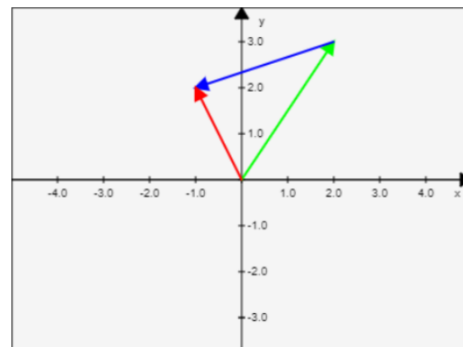
Draw the first number in green, moving the starting point.

Draw the second number in blue. Then move the starting point back to the origin.

Draw the sum in red from the origin.

Since this is a mathematical example, of course *Hilberto's* contribution must be adequately acknowledged by his co-representation.

```
configure sprite thisSprite as a MathPad
width: 400 height: 300 color: 245 245 245
set MathPad properties lineWidth: 1 onlyPoints: ☒
dimension: 2 maxValue: 5 startPoint: 0 0 0
on thisSprite
plot complex-number complex 2 + 3 * i color: 0 255 0
on MathPad thisSprite Change startpoint? ☒
plot complex-number complex -3 + -1 * i color: 0 0 255
on MathPad thisSprite Change startpoint? ☒
set MathPad properties lineWidth: 3 onlyPoints: ☒
dimension: 2 maxValue: 5 startPoint: 0 0 0
on thisSprite
plot complex-number
complex complex 2 + 3 * i + complex -3 + -1 * i color:
255 0 0
on MathPad thisSprite Change startpoint? ☒
```



6.2 Affine transformation of a triangle in \mathbb{R}^2

We define a triangle by its vectors. Then we define three points in the plane and three image points to which these are to be mapped.

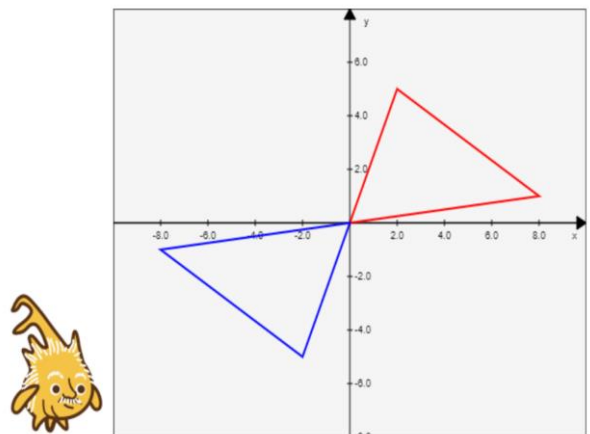
Then we create a two-dimensional *MathPad*, change the maximum value of the axes and draw the triangle in red. To its position vectors we apply the affine transformation and draw the result in blue. Since coordinate transformations are something for physics, Alberto presents the result.

```

set triangle to list list 0 0 list 8 1 list 2 5
set sourcePoints to list list 0 0 list 0 1 list 1 0
set targetPoints to list list 0 0 list 0 -1 list -1 0

configure sprite thisSprite as a MathPad
width: 400 height: 300 color: 245 245 245
set MathPad properties lineWidth: 2 onlyPoints? 
dimension: 2 maxValue: 10 startPoint: 0 0 0
on thisSprite
plot object of triangle color: 255 0 0
on MathPad thisSprite Change startpoint? 
set image to affine transformation of triangle
by sourcePoints --> targetPoints for MathPad
plot object of image color: 0 0 255
on MathPad thisSprite Change startpoint?

```



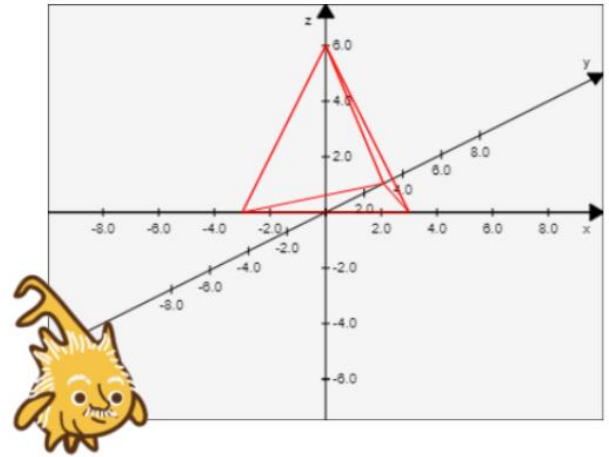
6.3 Rotation of a pyramid in R^3

First, of course, we want to draw a pyramid. We define the base and the top by location vectors.

We have them drawn by first drawing the base area, then lines from its corners to the top.

```
configure sprite thisSprite as a MathPad
width: 400 height: 300 color: 245 245 245
plot object-of base color: 255 0 0
on MathPad thisSprite Change startpoint? x
set MathPadProperty startPoint of thisSprite to list -3 0 0
plot line-to top color: 255 0 0
on MathPad thisSprite Change startpoint? ✓
plot line-to list 3 0 0 color: 255 0 0
on MathPad thisSprite Change startpoint? ✓
plot line-to list 0 3 0 color: 255 0 0
on MathPad thisSprite Change startpoint? ✓
plot line-to top color: 255 0 0
on MathPad thisSprite Change startpoint? ✓
```

```
set base to list list -3 0 0 list 3 0 0 list 0 3 0
set top to list 0 0 6
```

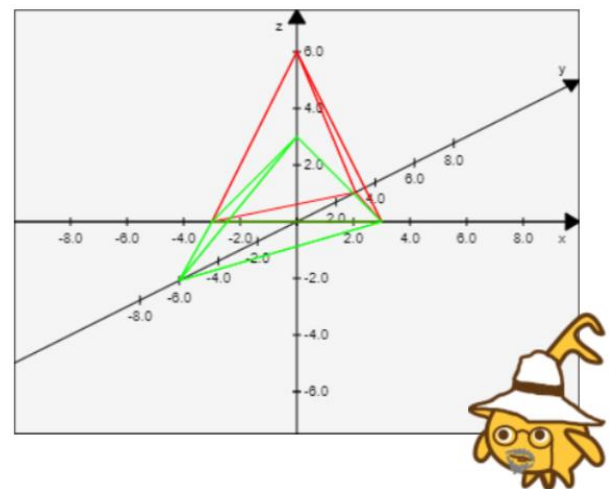


For rotations around the axes we need the three rotation matrices D_x , D_y and D_z . For a rotation around the x-axis by 90° we can apply the matrix directly to the base surface. Then we let draw the side lines to the rotated point by multiplying the rotation matrix with the transposed location vectors of the points. Since we - mathematically correct - multiply matrices with column vectors and get column vectors as results, we have to transpose them again to get normal location vectors.

So in total:

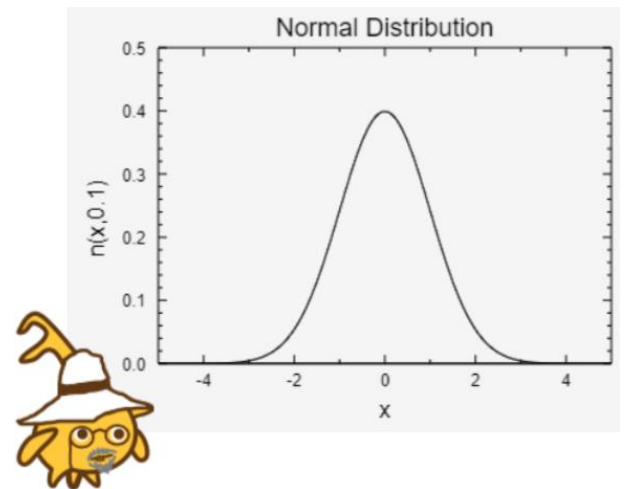
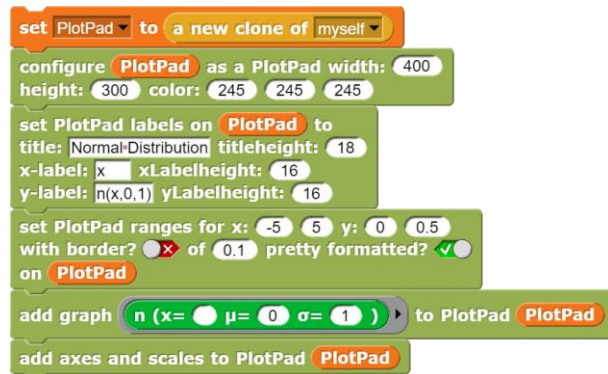
```
plot object-of apply Dx to points base color: 0 255 0
on MathPad thisSprite Change startpoint? x
set MathPadProperty startPoint of thisSprite to
linear operation Dx transpose list 3 0 0
plot line-to transpose linear operation Dx transpose top
color: 0 255 0
on MathPad thisSprite Change startpoint? ✓
plot line-to transpose linear operation Dx transpose list 3 0 0 color:
0 255 0
on MathPad thisSprite Change startpoint? ✓
plot line-to transpose linear operation Dx transpose list 0 3 0 color:
0 255 0
on MathPad thisSprite Change startpoint? ✓
plot line-to transpose linear operation Dx transpose top
color: 0 255 0
on MathPad thisSprite Change startpoint? ✓
```

```
set alpha to 90
set Dx to
matrix of vectors
list 1 0 0 list 0 cos of alpha neg of sin of alpha
list 0 sin of alpha cos of alpha
set Dy to
matrix of vectors
list cos of alpha 0 sin of alpha list 0 1 0
list neg of sin of alpha 0 cos of alpha
set Dz to
matrix of vectors
list cos of alpha neg of sin of alpha 0
list sin of alpha cos of alpha 0 list 0 0 1
```



6.4 Graph of normal distribution

Using the *PlotPad*, the graph of the normal distribution is drawn. To do this, we create a clone of *Hilberto*, configure it as *PlotPad* and create the graph.



6.5 Cartesian product of three sets

First of all, we create three sets with names, possible ages and occupations:

```
set names to set of { Hansen Peterson Anderson Carlsen }
set ages to set of { 34 25 46 51 }
set professions to set of { teacher bricklayer retailer lawyer }
```

From these sets we can now form the Cartesian product of names, ages, and occupations:

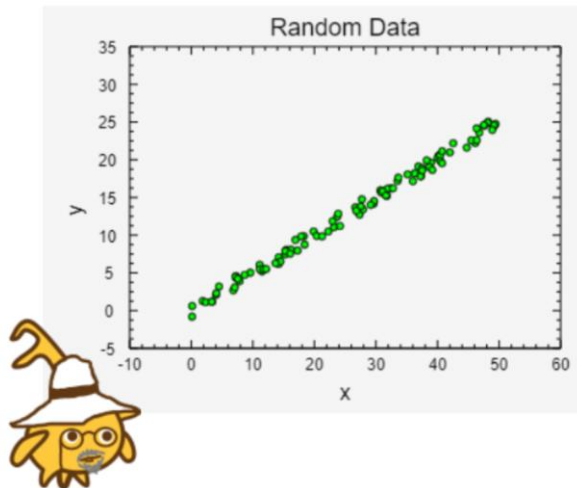
64	A	B	C
1	Hansen	34	teacher
2	Hansen	34	bricklayer
3	Hansen	34	retailer
4	Hansen	34	lawyer
5	Hansen	25	teacher
6	Hansen	25	bricklayer
7	Hansen	25	retailer
8	Hansen	25	lawyer
9	Hansen	46	teacher
10	Hansen	46	bricklayer
11	Hansen	46	retailer
12	Hansen	46	lawyer
13	Hansen	51	teacher
14	Hansen	51	bricklayer
15	Hansen	51	retailer
16	Hansen	51	lawyer
17	Peterson	34	teacher
18	Peterson	34	bricklayer
19	Peterson	34	retailer
20	Peterson	34	lawyer
21	Peterson	25	teacher
22	Peterson	25	bricklayer

```
item 3 of names x ages x professions
```

And thus nothing stands in the way of a transition to the topic of "relational databases", for example.

6.6 Representation of a set of points and the regression line

Using the *Data tools*, we create 100 random points scattering around a straight line with gradient $m=0.5$ and intercept $b=0$. We plot the obtained points in a diagram.



```

set points to 100 random points near a straight x-range 0 50
gradient 0.5 y-axis-intercept 0 range 2

configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245

set PlotPad labels on thisSprite to
title: RandomData titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16

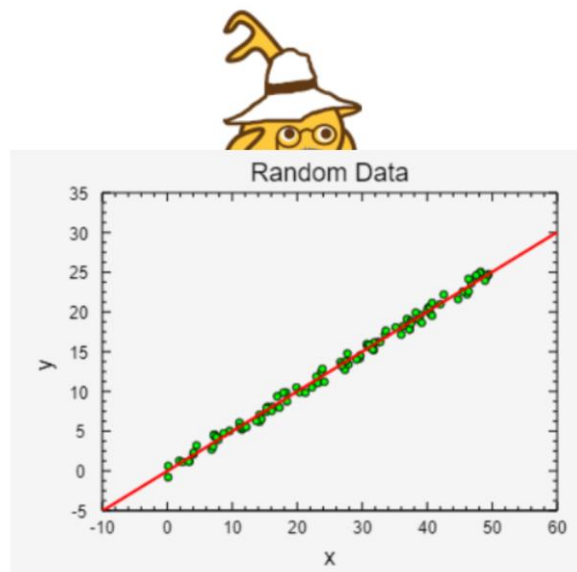
set PlotPad line properties style: continuous
width: 1 color: 0 0 0 on thisSprite

set PlotPad marker properties style: o circle width: 5
color: 0 255 0 connected? x on thisSprite

set PlotPad ranges for x: 0 50 y: 0 30
with border? ✓ of 0.1 pretty formatted? ✓
on thisSprite

add dataplot of numeric data: points to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
  
```

In addition, the regression line is also drawn.



```

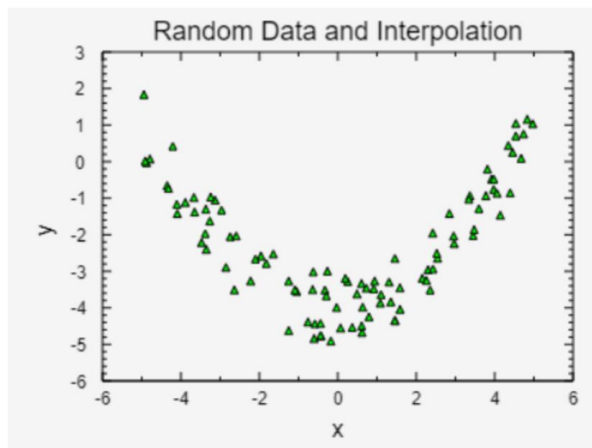
set PlotPad line properties style: continuous
width: 2 color: 255 0 0 on thisSprite

add graph regression line parameters of points to PlotPad thisSprite
  
```

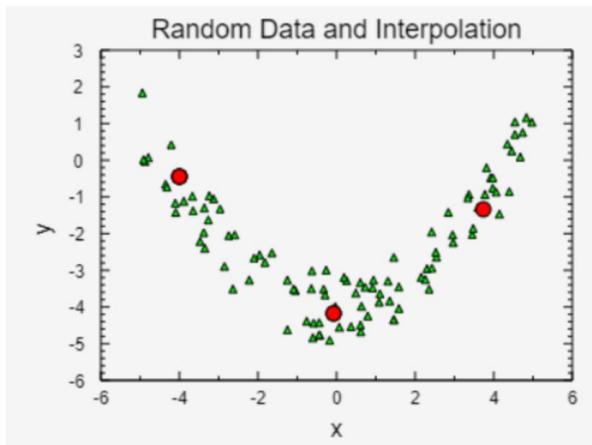
6.7 Interpolation polynomial through n points

We want to generate a data set of 100 points that scatter around a given function. We display these points in a diagram. We then select three points by clicking on the corresponding locations with the mouse and placing a red marker there. After that we draw an interpolation polynomial in red through these three points. Since this is a "mathematical" project, *Hilberto* is responsible for it.

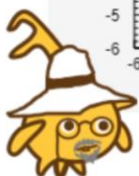
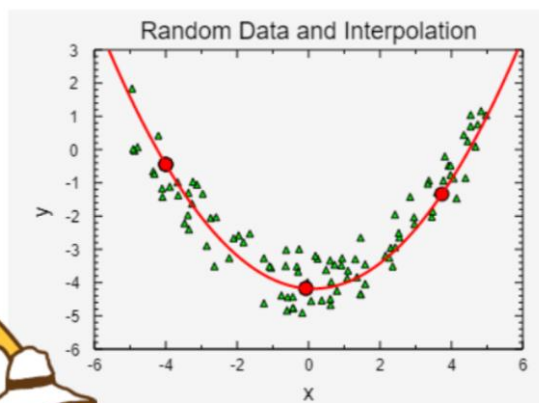
First of all, the random points:



Now we select the three points and plot them in the diagram right away.



And last we add the interpolation polynomial.



```
set data to 100 random points near 0.2 x ^ 2 - 4
between -5 and 5 range 2
```

```
configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245

set PlotPad labels on thisSprite to
title: Random-Data-and-Interpolation titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16

set PlotPad marker properties style: triangle width: 5
color: 0 255 0 connected? on thisSprite

get ranges for PlotPad thisSprite
from data with border 0.1

set pretty ranges on PlotPad thisSprite

add dataplot of numeric data: data to PlotPad thisSprite

set n to 0 and scales to PlotPad thisSprite
set points to list

set PlotPad line properties style: continuous
width: 1 color: 0 0 0 on thisSprite

set PlotPad marker properties style: circle width: 5
color: 255 0 0 connected? on thisSprite

repeat until n = 3
wait until mouse down?
add PlotPad graph-coordinates on thisSprite by mouse to points
add dataplot of numeric data: points to PlotPad thisSprite
change n by 1
wait 0.5 secs
```

```
set PlotPad line properties style: continuous
width: 2 color: 255 0 0 on thisSprite

add graph polynomial interpolation for points points to PlotPad
thisSprite
```

Tasks:

1. a: Generate "point clouds" scattering around other polynomials.
b: As in the example, specify some points in these clouds through which an interpolation polynomial is to be drawn.
c: Let draw these polynomials.
2. a: Experiment with the number of points you select. Do the results get better when you select more points?
b: Generate "point clouds" that scatter around non-rational function graphs (trigonometric, ...). Can you also describe these by interpolation polynomials?
c: Formulate a rule for when and how to use interpolation polynomials in a meaningful way - and why just so.

6.8 Approximation of a tangent by secants

We want to show that a sequence of secant slopes converges against the gradient of the tangent line at a point. To do this, we configure a sprite called *PlotPad* as *PlotPad* and draw the graph of a function, here:

$$y = 0,3 \cdot x^3 - 3 \cdot x.$$

We want to draw a sequence of secants near the right minimum, which approach the tangent. Of course we need a point $(x_0|y_0)$ near the minimum to do this:

```
set x0 to 1.7
set y0 to 0.3 * x0 ^ 3 - 3 * x0
```

We may as well draw it.

```
set PlotPad line properties style: continuous
width: 1 color: 0 0 0 on PlotPad
set PlotPad marker properties style: circle width: 5
color: 255 0 0 connected? x on PlotPad
add dataplot of numeric data: list list x0 y0 to PlotPad
```

As sequence to approach the point we choose $a_n = \frac{2}{n}$, and we let generate the first 20 elements.

```
set sequence to first 20 elements of sequence 2 /
```

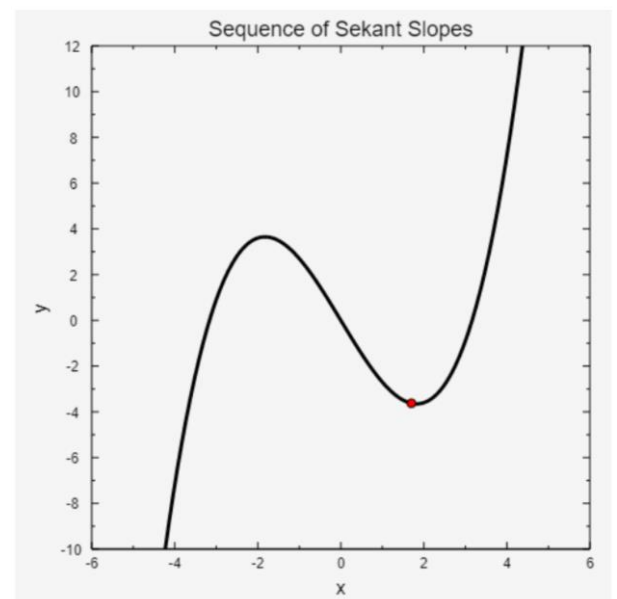
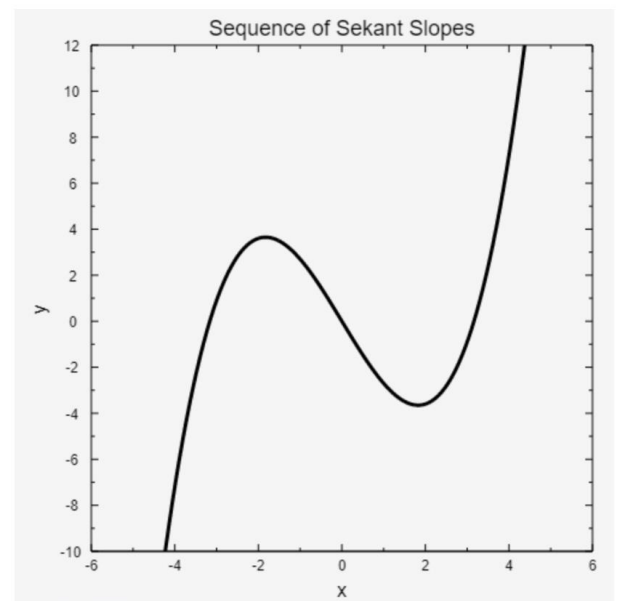
The rest is also simple: we have the secant slopes calculated ...

```
set secantSlopes to
sequence of secant slopes for 0.3 * x ^ 3 - 3 * x
at x0 calculated with sequence sequence
```

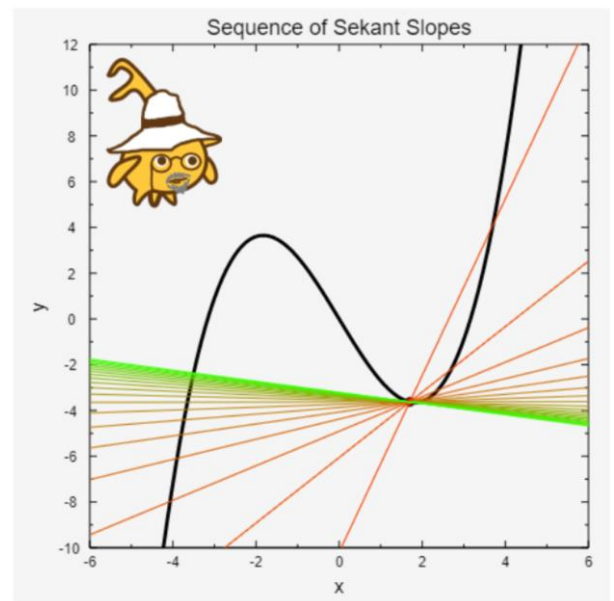
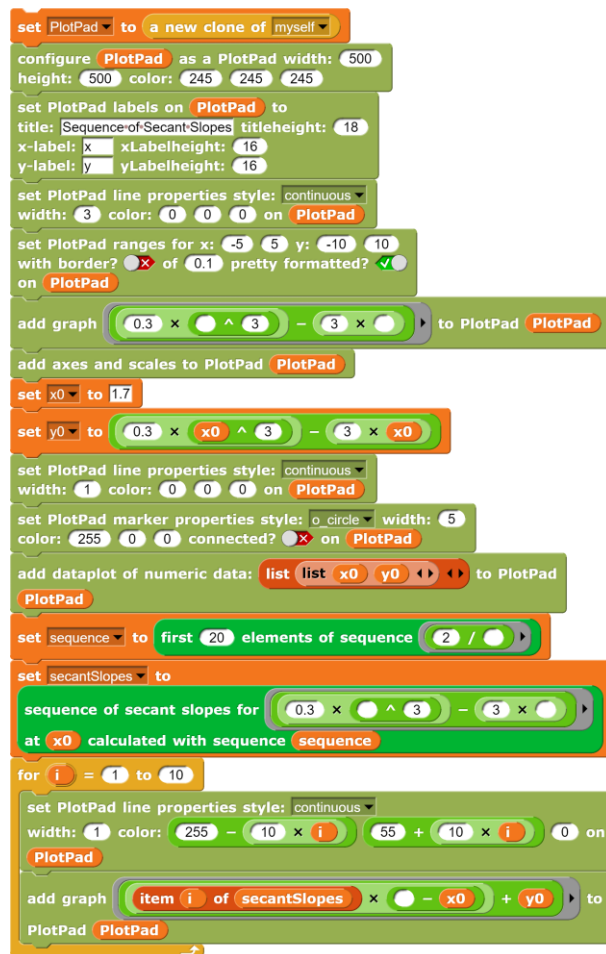
... and the secants drawn in color gradations.

```
for i = 1 to 10
set PlotPad line properties style: continuous
width: 1 color: 255 - 10 * i 55 + 10 * i 0 on
PlotPad
add graph item i of secantSlopes * x - x0 + y0 to
PlotPad PlotPad
```

```
set PlotPad to a new clone of myself
configure PlotPad as a PlotPad width: 500
height: 500 color: 245 245 245
set PlotPad labels on PlotPad to
title: Sequence of Secant Slopes titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16
set PlotPad line properties style: continuous
width: 3 color: 0 0 0 on PlotPad
set PlotPad ranges for x: -5 5 y: -10 10
with border? x of 0.1 pretty formatted? y
on PlotPad
add graph 0.3 * x ^ 3 - 3 * x to PlotPad PlotPad
add axes and scales to PlotPad PlotPad
```



The whole thing - with result - once again in one piece:



Tasks:

1. In addition, have the "correct" tangent drawn in the diagram.
2. Choose as sequence for the secant calculation other sequences approaching the point $(x_0|y_0)$ from the other or both sides.
3. a: Similarly, illustrate the calculation of roots using the Newton method.
b: Select some cases where the procedure works well or hardly or not at all.

6.9 Finite series

We want to approximate some of the common mathematical constants and functions via series expansions - and also try out how long you actually have to calculate to get good results.


Let's start with π . In a compendium of formulas¹⁷ or on Wikipedia¹⁸ we can find a formula for calculating π : the *Leibniz-series*:

$$\frac{\pi}{4} = \sum_{i=0}^n \frac{(-1)^i}{(2 \cdot i + 1)}$$

We can implement it directly in *SciSnap!*. But when is the result actually "good"?

To answer this question we create a table.

table			
7	A	B	C
1	k	10^k	Pi
2	1	10	3.232315809405594
3	2	100	3.1514934010709914
4	3	1000	3.1425916543395442
5	4	10000	3.1416926435905346
6	5	100000	3.1416026534897203
7	6	1000000	3.1415936535887745



Actually, it is strange that one must calculate so long for so little improved accuracy. You might do something like that in rainy Hanover in 1673 to avoid having to go for a walk - but now? We just try the *BIGNUM* library from *Snap!* for exact calculations with *Scheme numbers*. Then the calculation takes even longer and we get amazing results that don't look like π .

After a long search, we discover the fraction bar in the middle, although it is no longer visible in the second line. *Scheme numbers* are exact fractions and not floating-point numbers. We therefore have the exact *Scheme numbers* converted to inexact floating point representation before we enter them into the table.


```
set Pi to 4 x Σ (-1 ^ i / (2 x i + 1) i = 0 to n
```

```
set table to new 3 by 0 table with labels: k 10^k Pi
for k = 1 to 6
  set Pi to 4 x Σ (-1 ^ i / (2 x i + 1) i = 0 to n
  add row list k 10 ^ k Pi to table
```

With a million summands you have to wait a bit for the result! 😊

```
USE BIGNUMS ✓
set table to new 3 by 0 table with labels: k 10^k Pi
for k = 1 to 3
  set Pi to 4 x Σ (-1 ^ i / (2 x i + 1) i = 0 to n
  add row list k 10 ^ k Pi to table
```

table			
4	A	B	C
1	k	10^k	Pi
2	1	10	47028692/14549535
3	2	100	830451968305093031586835172847858137121823705761010747562
4	3	1000	463771016605518999270845997524183318155107079951977133705



```
add row list k 10 ^ k Scheme number inexact of Pi to table
```

table			
4	A	B	C
1	k	10^k	Pi
2	1	10	3.2323158094055926
3	2	100	3.15149340107099
4	3	1000	3.1425916543393395

¹⁷ Ask your grandpa what it is. 😊

¹⁸ https://en.wikipedia.org/wiki/Leibniz_formula_for_pi

Despite the effort, the results are almost the same as before. Thus, the poor convergence behavior of the series is not due to the inaccuracies of the standard arithmetic of a programming language (here: *JavaScript*), but to its structure. Good to know! 😊

Tasks:

1. Find out the meaning of the terms "*scheme numbers*" and "*floating point numbers*".
2. Look for other series expansions for π , which converge better than the Leibniz series.
3. Find and implement a series expansion for Euler's number e .
4. Inform yourself about reasons to calculate with the accuracy of Scheme numbers instead of that for floating point numbers.
5. Write scripts for the series expansion of trigonometric functions, e.g. $\sin(x)$, $\cos(x)$, ... Note that the angle must be specified in radians.

6.10 Application of the Taylor series to the mathematical pendulum

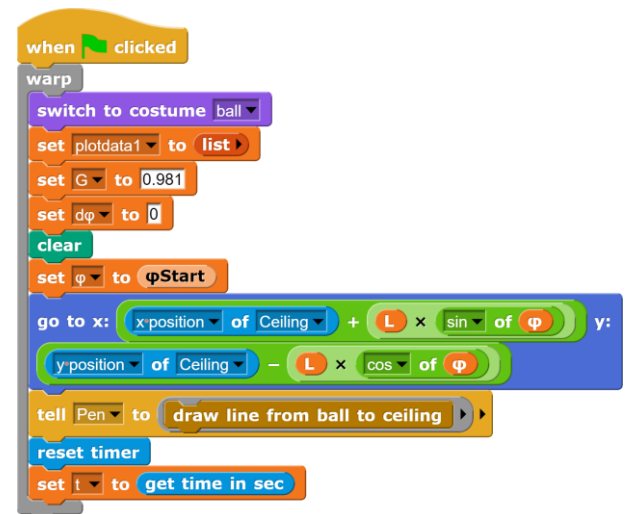
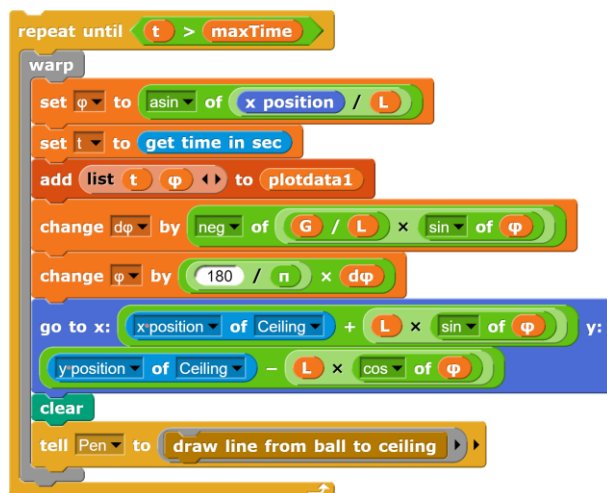
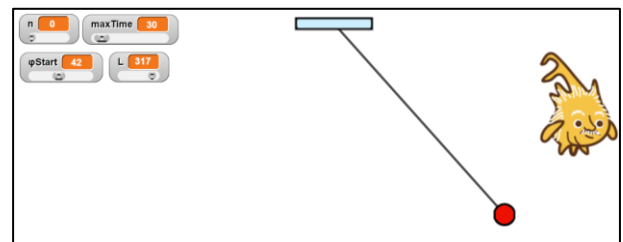
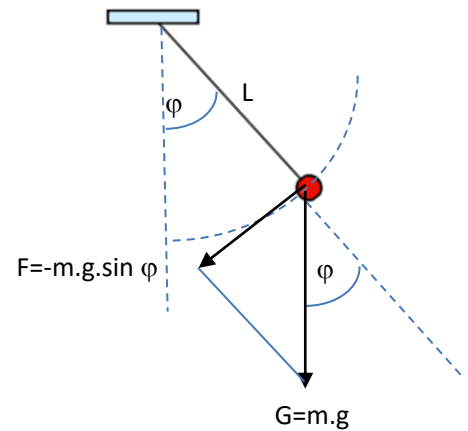
A very illustrative application of series expansions is the simulation of a mathematical pendulum, i.e. a string pendulum. Usually, one works there with the approximation that for small angles the value of the sine (in radians) corresponds approximately to the value of its argument - i.e. one breaks off the series expansion of the sine function after the first summand. Let's take a closer look: We get the force F as $F = -G \cdot \sin \varphi = -m \cdot g \cdot \sin \varphi$ accelerating the ball on the circular path. According to the basic equation of mechanics, this force is equal to the inertial force $m \cdot \ddot{s} = m \cdot a$. If we use the relation for the angle in radians $\varphi = \frac{s}{L}$, we get

$$\ddot{\varphi} = -\frac{g}{L} \cdot \sin \varphi$$

For small angles the approximate $\sin \varphi \approx \varphi$ is valid and thus

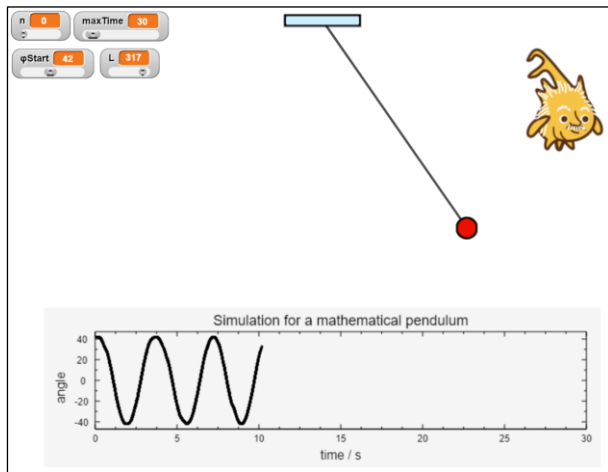
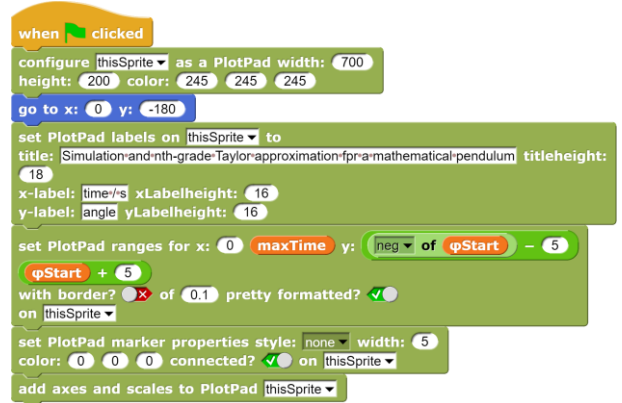
$$\ddot{\varphi} \approx -\frac{g}{L} \cdot \varphi$$

Let's see if this works! First of all we simulate the "real" pendulum movement by taking the acceleration from the current position, from it we determine the change of the velocity and from it again the new position. We put the data into a list *plotdata1*. Some further quantities like the length of the pendulum and the initial displacement are given by variables in the slider view. First of all, we plot the initial situation: A pen draws required lines, and the pendulum hangs on the ceiling of the lab, of course.



Now we start: We measure the current deflection and the time and write both values into the list of measured values. Then we calculate the current acceleration, which changes the angular velocity, and from that the new angle. After that we let draw the new situation. And this again and again.

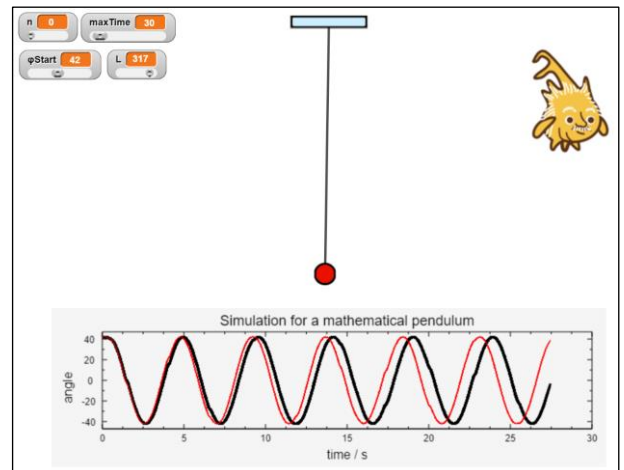
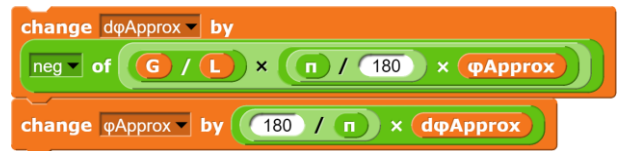
To this arrangement we add another sprite as *PlotPad*: the *Plotter*. It is fast enough to display the current data in real time. Therefore we insert the block for it into the simulation loop of the pendulum.



Alberto is visibly thrilled that this is working out so well!

But the real goal was to see how good the approximation is. For this purpose we introduce a second data list *plotdata2* as well as an "approximated" angle ϕ_{Approx} and the corresponding angular velocity. The "real" angle ϕ and the approximated one start with the same value. Then we measure the "real" deflection of the pendulum and calculate the approximated value. We draw both in the diagram - the calculated value in red.

You can see that the approximation is quite good at the beginning, but diverges with the real-time values as time goes on.



This can be done better!

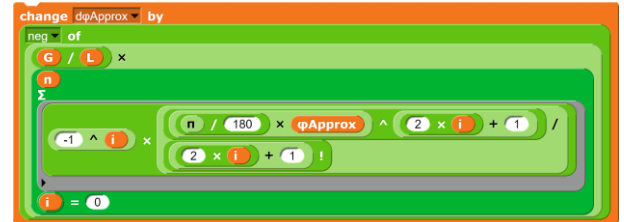
Instead of the linear approximation, we choose the Taylor series of the sine, of which we use n sums as approximation. We specify n as slider variable.

$$\sin \varphi = \sum_{i=0}^n (-1)^i \cdot \frac{\varphi^{2i+1}}{(2i+1)!} = \varphi - \frac{\varphi^3}{3!} + \frac{\varphi^5}{5!} - \dots$$

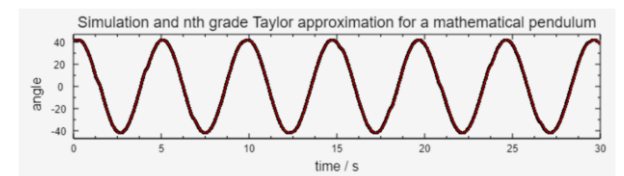
We can copy this directly into *SciSnap*!:



Together with the conversion of angles into radians we obtain for the angular acceleration:



Already with an extension of the approximation by one series element (i.e. $-\frac{\varphi^3}{3!}$) the deviation from the measured values are no longer visible in the diagram.



Tasks:

1. Let the simulation run longer and notice when - depending on the number of summands of the Taylor series - deviations appear.
2. Do the same for different start angles of the pendulum.

6.11 Fourier expansion for a square wave signal with numerical integration

One way to represent a 2π -periodic function $f(x)$ is the Fourier representation

$$f(x) \cong a_0 + \sum_{k=1}^{\infty} (a_k \cdot \cos(k \cdot x) + b_k \cdot \sin(k \cdot x)) \quad a_0 = \frac{1}{2\pi} \cdot \int_0^{2\pi} f(x) \cdot dx$$

$$a_k = \frac{1}{\pi} \cdot \int_0^{2\pi} f(x) \cdot \cos(k \cdot x) \cdot dx \quad b_k = \frac{1}{\pi} \cdot \int_0^{2\pi} f(x) \cdot \sin(k \cdot x) \cdot dx; \quad k = 1, 2, 3, \dots$$

We want to determine the coefficients for the series expansion of a rectangular signal purely numerically and test how the length of the series expansion affects the accuracy of the results.

First of all we need a square wave signal with period 2π . A function that returns this, would be e.g.

$$f(x) = \begin{cases} -1 & \text{if } (x \bmod 2\pi - \pi) > 0 \\ 1 & \text{if } (x \bmod 2\pi - \pi) < 0 \\ 0 & \text{otherwise} \end{cases}$$

Of course, let's look at this in the diagram before we think it is a square wave signal:

```

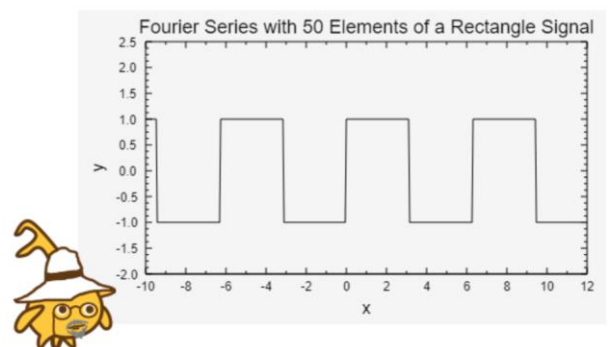
set n to 50
configure PlotPad as a PlotPad width: 500
height: 300 color: 245 245 245
set PlotPad labels on PlotPad to
title: join FourierSeriesWith n Elements of a Rectangle Signal titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16
set PlotPad ranges for x: -10 10 y: -2 2
with border? of 0.1 pretty formatted?
on PlotPad
add graph f to PlotPad PlotPad
add axes and scales to PlotPad PlotPad

```

```

set f to
if mod 2 x n - n > 0 then -1 else
if mod 2 x n - n < 0 then 1 else 0

```



True so. 😊

Now, for each value of x , we need to calculate the first n coefficients of the Fourier series. For this we use the block for numerical integration.

```

2
∫ 1 dx
1
calculated with 100 intervals

```

So, let's try it for $a_0 = \frac{1}{2\pi} \cdot \int_0^{2\pi} f(x) \cdot dx$. Since we already have the "ringified term" for the function f , we can simply write off:

```

set a0 to
1 / 2 x n x ∫ f dx
0
calculated with 500 intervals

```

For the other coefficients we have to complete the trigonometric terms and obtain for

a_k :

```

1 / n x
2 x n
if mod 2 x n - n > 0 then
neg of cos of i x 180 / n else
if mod 2 x n - n < 0 then
cos of i x 180 / n else 0
0
calculated with 100 intervals

```

and b_k :

```

1 / n x
2 x n
if mod 2 x n - n > 0 then
neg of sin of i x 180 / n else
if mod 2 x n - n < 0 then
sin of i x 180 / n else 0
0
calculated with 100 intervals

```

These terms now only have to be summed up:

This gives us the following script in total.

```

Fourier series of f(x) with n = 10 elements
script variables a0 ak bk result
warp
set a0 to 1 / (2 * n) * ∫ f dx
  calculated with 500 intervals
set ak to list
set bk to list
for i = 1 to n
  add f
    1 / n *
    2 * n
    if mod(2 * n - n) > 0 then
      neg of cos of i * x * 180 / n
    else
      if mod(2 * n - n) < 0 then
        cos of i * x * 180 / n
      else 0
    calculated with 100 intervals
  ak
  add f
    1 / n *
    2 * n
    if mod(2 * n - n) > 0 then
      neg of sin of i * x * 180 / n
    else
      if mod(2 * n - n) < 0 then
        sin of i * x * 180 / n
      else 0
    calculated with 100 intervals
  bk
set result to a0
for i = 1 to n
  change result by
    item i of ak * cos of i * x * 180 / n +
    item i of bk * sin of i * x * 180 / n
report result
  
```

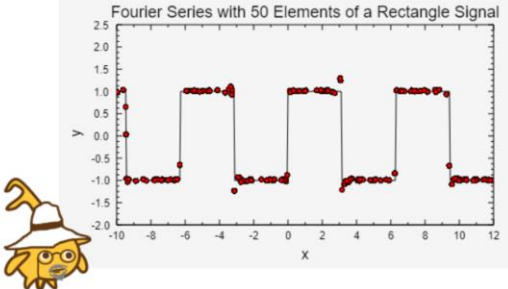
```






set result to a0
for i = 1 to n
  change result by
    item i of ak * cos of i * x * 180 / n +
    item i of bk * sin of i * x * 180 / n
report result
  
```

We have two "screws" in it, which we can turn: on the one hand the number n of the terms of the Fourier series can be changed, on the other hand the number i of the intervals in the numerical integration. The effects of changes can be easily observed if we randomly draw values from the range of values of x and plot the result together with the function f . Deviations then show up immediately.

```

set n to 50
set data to list
set PlotPad marker properties style: o circle width: 5
color: 255 0 0 connected? x on PlotPad
forever
  set x to random * 22 - 10
  add list x Fourier series of f(x) with n elements to data
  add dataplot of numeric data: data to PlotPad PlotPad
  
```

n summands	i intervals	result	comment
50	100		Actually quite good except for "slips" that are at the jump points of the function.

50	50		That should be clear.
50	200		Better, but only slightly better than with 100 summands.
25	200		Worse, but only marginally worse than with 50 summands.
10	100		Perhaps quite a good compromise between accuracy and computation time requirements.
5	75		In many cases, this is probably still acceptable.

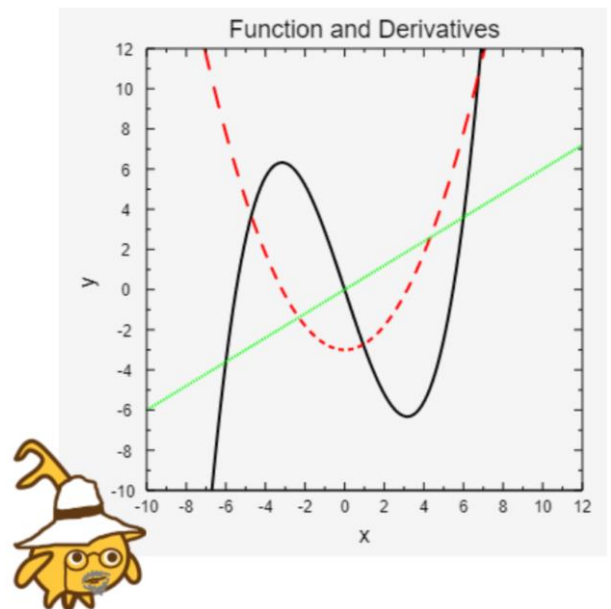
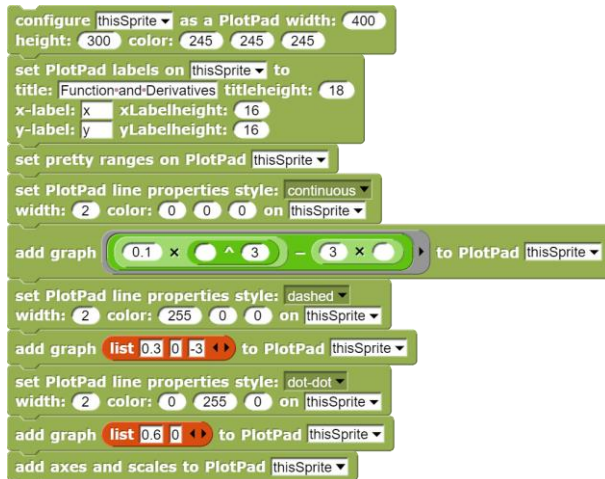
You can see that the accuracy required depends on the task. For example, are slips at the jump points acceptable or is it precisely there that precision is important? It can also be seen that very different effects can be achieved with the same effort.

Tasks:

1.
 - a: Give function terms for a triangular signal, a signal made of peaks, Have the function graphs drawn.
 - b: Calculate the Fourier series for the signals.
 - c: Experiment as shown to find a reasonable tradeoff between accuracy and computation time requirements.
2.
 - a: Create tables with the data for a triangle signal, a signal from peaks, ... using the function terms.
 - b: Output the signals through the speaker.
 - c: Generate the corresponding tables using Fourier series of different quality and listen to them as sounds as well. Do you perceive any differences?
3. Find out about applications of Fourier series.

6.12 Drawing a function and its derivatives

We want to use a *PlotPad* to display a function and its first two derivatives in different colors and line styles. To do this, we create a new sprite, switch to its script area, and configure it appropriately. The function terms are specified once as a "ringified" operator, then twice as a list of polynomial coefficients.



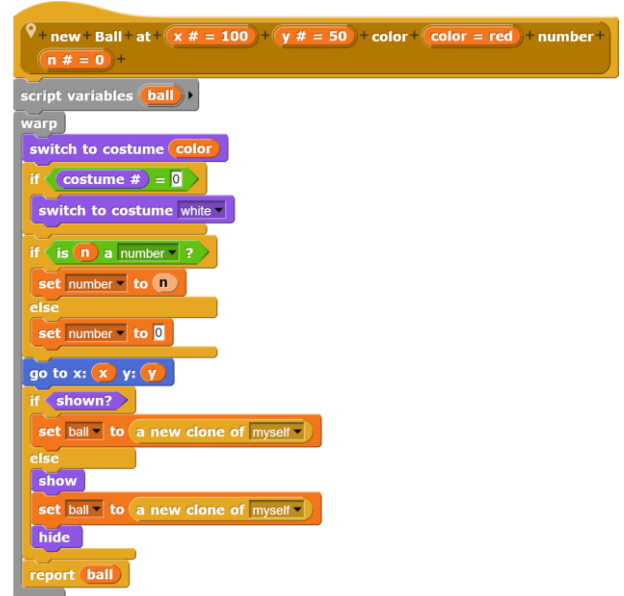
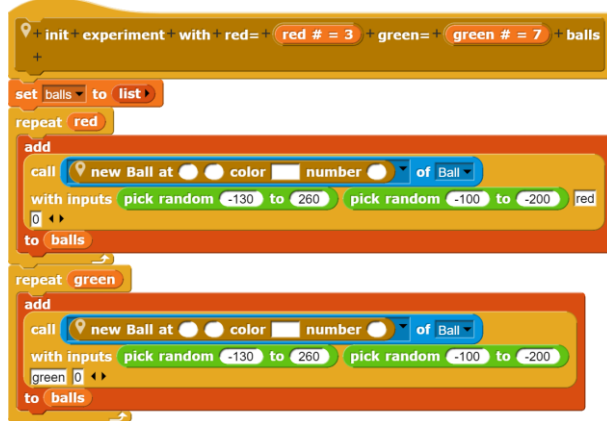
Tasks:

1. Plot different types of functions (trigonometric functions, logarithms, polynomials, ...) as graphs on a *PlotPad*.
2. Complete the function graphs with their derivatives.
3. Select different ranges of values, precision of the number representation and text sizes and labels for the representations.

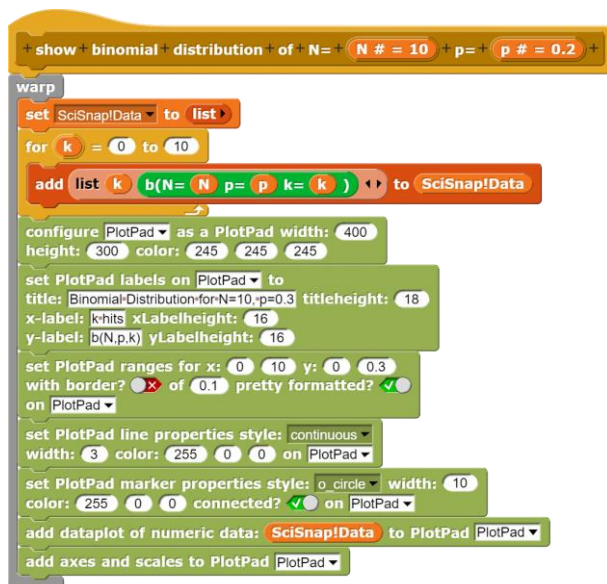
6.13 Random experiments on the binomial distribution

Since the work here is really mathematical, we ask *Gundolf de Jong*, the talented young mathematician, for his help. On the one hand, he is to perform random experiments, on the other hand, he is to evaluate the experiments and compare them with the corresponding binomial distribution. To do this, of course, he generates a diagram.

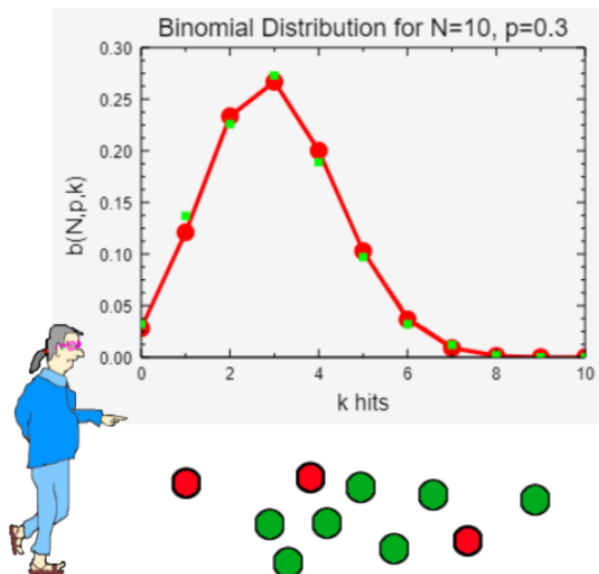
For the experiments, *Gundolf* needs red and green balls that can be numbered. These are supplied to him by a sprite called *Ball*, which can be asked for appropriate copies.



After the execution of `init experiment with red= 3 green= 7 balls` Gundolf is still quite alone in the room with his balls. He therefore begins to experiment: from his 10 balls he randomly takes one and checks its color. If it is red, he counts the ball and puts it back. He does this 10 times each and calls the whole thing an experiment. He performs these experiments n times and divides the number of red balls drawn by the total number of experiments. He returns the result.



After that, he can draw a plot on a Sprite *PlotPad* showing the corresponding binomial distribution in red. He then draws the result of $n=1000$ experiments in green on top.



```

init experiment with red= 3 green= 7 balls
show binomial distribution of N= 10 p= 0.3
set PlotPad marker properties style: square width: 5
color: 0 255 0 connected? ☒ on PlotPad
set PlotPad line properties style: continuous
width: 1 color: 0 255 0 on PlotPad
add dataplot of numeric data: result of 1000 experiments to PlotPad
PlotPad

```

Tasks:

1. Change the rules of the game: Do not put the picked balls back, change the number of balls in total, per color or related to the number of drawn balls. Define what counts as a "success".
2. Plot the results on graphs. Find out about the expected distribution and enter it as well. Discuss deviations if necessary.

6.14 Fast Fourier Transformation (FFT)

The *FFT* (Fast Fourier Transform) is an efficient algorithm to calculate the Discrete Fourier Transform (*DFT*) of a vector. A discrete periodic frequency spectrum is assigned to the data. It is one of the standard methods, e.g. for scientific and technical applications. It is better to read about the details elsewhere before using the corresponding *SciSnap!* block. At this point we want to describe the procedure by a *Snap!* script and then demonstrate the possibilities of the *SciSnap!* block by some examples.

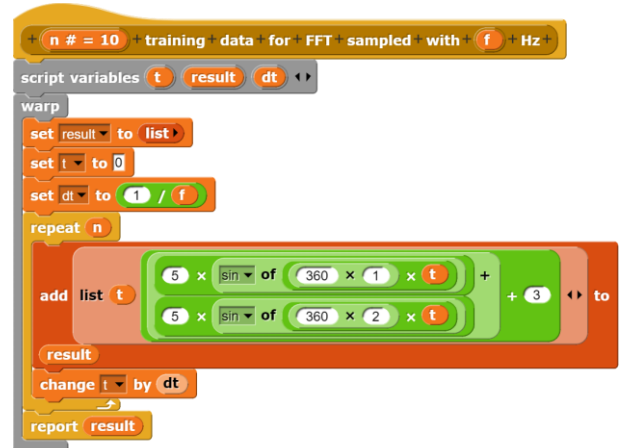
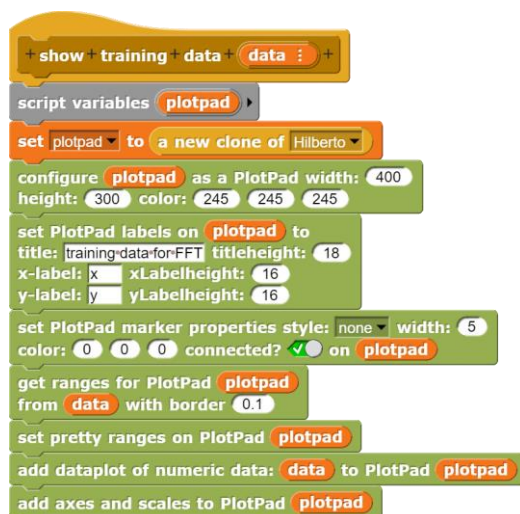
Let the shape of a function be given by a sequence of function values. We can imagine their generation as a reading of the function value in regular intervals, e.g. in the case of a time-dependent signal in certain time intervals. This is called *sampling*. However, only the function values, not the "interpolation points", go into the calculation of the FFT. This vector is now repeatedly halved by dividing it by even and odd indices until only single function values are left. From these the DFT is then generated by new combinations. Since this division is only possible for vector lengths of 2^N , a vector of other length must be completed before processing to this length by appending e.g. zeros. The algorithm then works as follows¹⁹:

```

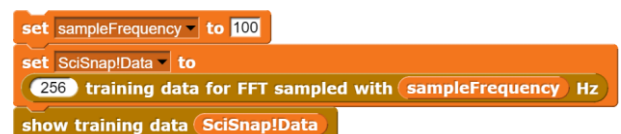
FFT(vector) //index starts with 0
  n = length of vector
  if n<=1 result = vector as complex number
  else halve vector into two vectors even and odd with even resp.
    odd indices and apply FFT to these:
    even = FFT(values with even index)
    odd = FFT(values with odd index)
    for m from 0 to n/2-1 do
      resultm = evenm+oddm*e-2πim/n
      resultn/2+m = evenm-oddm*e-2πim/n
  return result

```

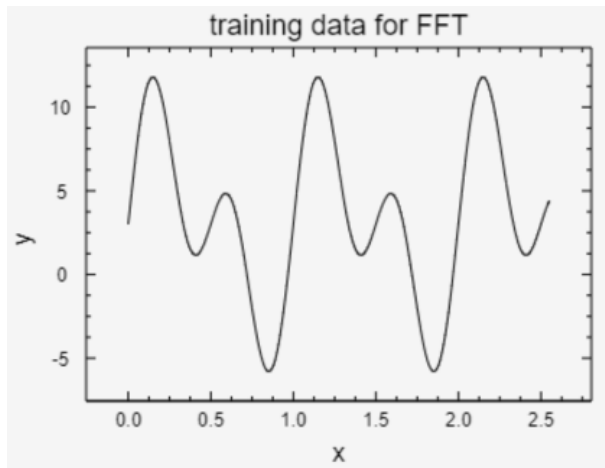
To illustrate the processes, we first need training data on which we can test the FFT. For the sake of simplicity, we superpose two sine functions and add the value 3. Let's take a look at the result.



We set the sample frequency to 100 Hz and determine the training data.



¹⁹ Source: Wikipedia https://de.wikipedia.org/wiki/Schnelle_Fourier-Transformation



So this is what the function looks like. To the second column of training data - the sampled function values - we apply the FFT algorithm as described. We can transfer it directly to *SciSnap!*.

FFT with SciSnap! blocks of vector

column 2 of SciSnap!Data with first item? ☒

```

+ FFT with SciSnap! blocks of vector + data : +
script variables evenPart oddPart evenResult oddResult n
warp
set n to length of data
if n ≤ 1
  report list complex item 1 of data + 0 * i
set evenPart to
  FFT with SciSnap! blocks of vector
  keep items index mod 2 = 1 input names: value index
  from data
set oddPart to
  FFT with SciSnap! blocks of vector
  keep items index mod 2 = 0 input names: value index
  from data
set evenResult to list
set oddResult to list
for k = 1 to n / 2
  add
    complex item k of evenPart +
    complex item k of oddPart -
    complex 1 * e^i ~360 × k - 1 / n
  to evenResult
  add
    complex item k of evenPart -
    complex item k of oddPart +
    complex 1 * e^i ~360 × k - 1 / n
  to oddResult
report append evenResult oddResult

```

If the vector passed has length one, it is returned as a complex number in SciSnap! format.

The data is divided into two parts with even and odd indices, respectively, to each of which the FFT is recursively applied.

The two halves of the result are put together as described ...

... and returned appended.

Das Ergebnis sehen wir uns für (hier: nur) 8 Funktionswerte an.

FFT with SciSnap! blocks of vector

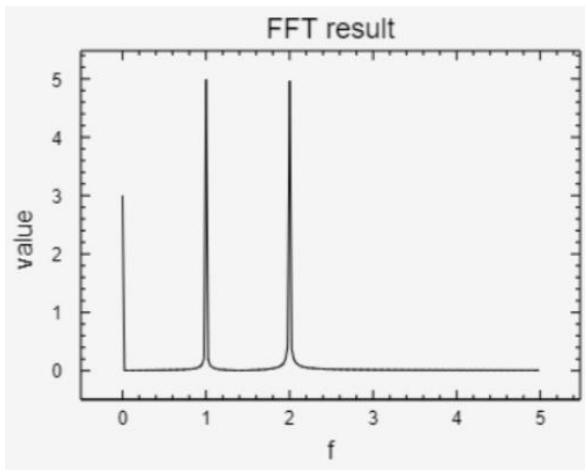
column 2 of SciSnap!Data with first item? ☒

	A	B	C
1	complexNumberCartesianStyle	48.96931370990699	0
2	complexNumberCartesianStyle	-3.8922516608711244	8.164322519753798
3	complexNumberCartesianStyle	-3.4812560903797785	3.334362009251132
4	complexNumberCartesianStyle	-3.4121839517945776	1.377837140911721
5	complexNumberCartesianStyle	-3.3979303038160324	0
6	complexNumberCartesianStyle	-3.4121839517945767	-1.377837140911721
7	complexNumberCartesianStyle	-3.4812560903797785	-3.334362009251132
8	complexNumberCartesianStyle	-3.8922516608711235	-8.164322519753798

But what should we do with it?

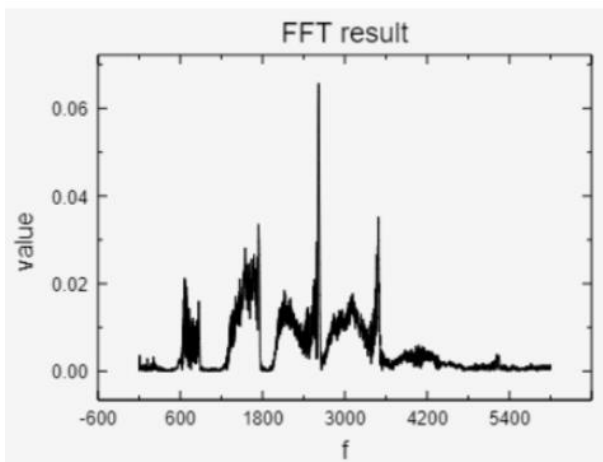
For a frequency spectrum, we are not interested in the total complex numbers, but (mostly) only their amounts. We obtain these, for example, by converting the numbers into the polar form and then splitting off the imaginary part - and the first column with the type at the same time. We scale these values in such a way that the amplitudes of the partial functions result correctly, and before these values we insert the associated frequencies.

With the result we can have the frequency diagram drawn. If we sample 4096 values with 100 Hz and run the procedure as described, we get a nice spectrum.



The absolute term with the value 3 as well as the sine functions of 1 Hz and 2 Hz with the amplitudes 5 are clearly visible.

Fourier transformations must also be reversible (*IFFT*). For this, the imaginary parts, the phases, are required after all. The *SciSnap!* block for FFTs therefore contains the three corresponding options. With its help we get the spectrum we are looking for much easier. And also the meowing from the Snap! sounds ("cat") can be analyzed quickly.



show fft data frequency_spectrum of samples of sound Cat sampled with sample rate of sound Cat Hz from fMin= 0 to fMax= 6000

```

set fftData to
  map complex polar style over
  column 2 of FFT with SciSnap! blocks of vector
  column 2 of SciSnap!Data with first item? ✓
  with first item? ✓

set length to length of fftData

set fftData to map 2 x / length over fftData

replace item 1 of fftData with item 1 of fftData / 2

set fftData to
  map
    report list index - 1 x sampleFrequency / length value
  over fftData
  input names: value index

show fft data fftData from fMin= 0 to fMax= 100

set sampleFrequency to 100

set SciSnap!Data to
  4096 training data for FFT sampled with sampleFrequency Hz

show training data SciSnap!Data

show fft data
  frequency_spectrum of column 2 of SciSnap!Data with first item? ✓
  sampled with sampleFrequency Hz
  from fMin= 0 to fMax= 5

```

frequency_spectrum of sampled with 100 Hz

frequency_spectrum
complex_FFTdata
IFFT_of_FFTdata

```

+ show+fft+data+ data : + from +fMin+= fMin # = 0 +to+fMax+=
fMax = 100 +

script variables plotpad plotData
warp
set plotData to empty table
for i = 1 to length of data
  if
    item 1 of item i of data ≥ fMin and
    item 1 of item i of data ≤ fMax
  add item i of data to plotData

set plotpad to a new clone of Hilberto
configure plotpad as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on plotpad to
  title: FFTresult titleheight: 18
  x-label: f xLabelheight: 16
  y-label: value yLabelheight: 16
set PlotPad marker properties style: none width: 5
color: 0 0 0 connected? ✓ on plotpad
get ranges for PlotPad plotpad
from plotData with border 0.1
set pretty ranges on PlotPad plotpad
add dataplot of numeric data: plotData to PlotPad plotpad
add axes and scales to PlotPad plotpad

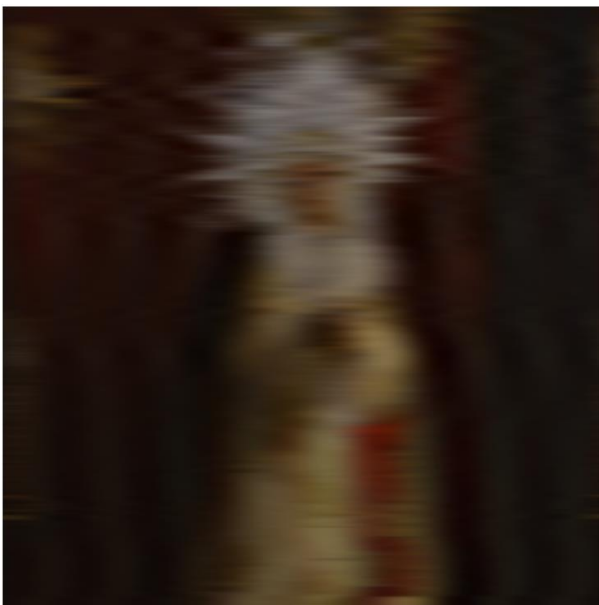
```


FFTs are not only used for sounds, but also for image processing. We take the three color channels of an image as vectors, apply the FFT operation in each case and cut off the higher frequencies, for example. The result is then composed of three color channels created by iFFT from the altered frequency values. The results are quite "modern". 😊

If you leave only 5% of the low frequencies, you can still see a surprising number of details. So the higher frequencies are responsible for the finer details.



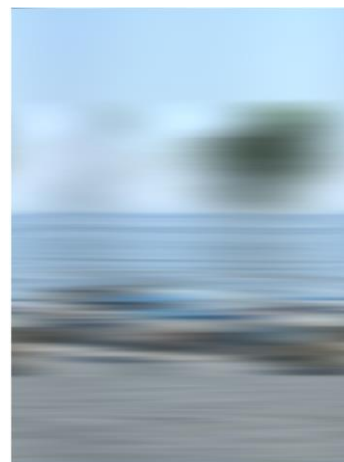
With 2% it becomes already more difficult: an "Easter picture".



```

script variables red green blue
switch to costume Indien
set sampleFrequency to 1
set SciSnap!Data to pixels of costume current
set red to
  complex_FFTdata of column 1 of SciSnap!Data with first item? ✓
  sampled with sampleFrequency Hz
set green to
  complex_FFTdata of column 2 of SciSnap!Data with first item? ✓
  sampled with sampleFrequency Hz
set blue to
  complex_FFTdata of column 3 of SciSnap!Data with first item? ✓
  sampled with sampleFrequency Hz
warp
set red to columns 2 3 of red from row 1 to last
set green to columns 2 3 of green from row 1 to last
set blue to columns 2 3 of blue from row 1 to last
for i = 1 to length of red
  if i > 0.02 × length of red
    replace item i of red with list 0 0
    replace item i of green with list 0 0
    replace item i of blue with list 0 0
set red to iFFT_of_FFTdata of red sampled with sampleFrequency Hz
set green to iFFT_of_FFTdata of green sampled with sampleFrequency Hz
set blue to iFFT_of_FFTdata of blue sampled with sampleFrequency Hz
set artPhoto to empty table
add column red to artPhoto
add column green to artPhoto
add column blue to artPhoto
add column column 4 of SciSnap!Data with first item? ✓ to artPhoto
switch to costume artPhoto
  
```

And at 1%? Have a guess! 😊



Tasks:

1. Generate different function waveforms from sine/cosine functions. Then experiment with different sample rates and numbers of sample points to generate frequency spectra. In each case, compare your results with the expected result - which you know.
2. Create spectra of different sounds, e.g. from instruments, tuning forks, speech, singing, ...

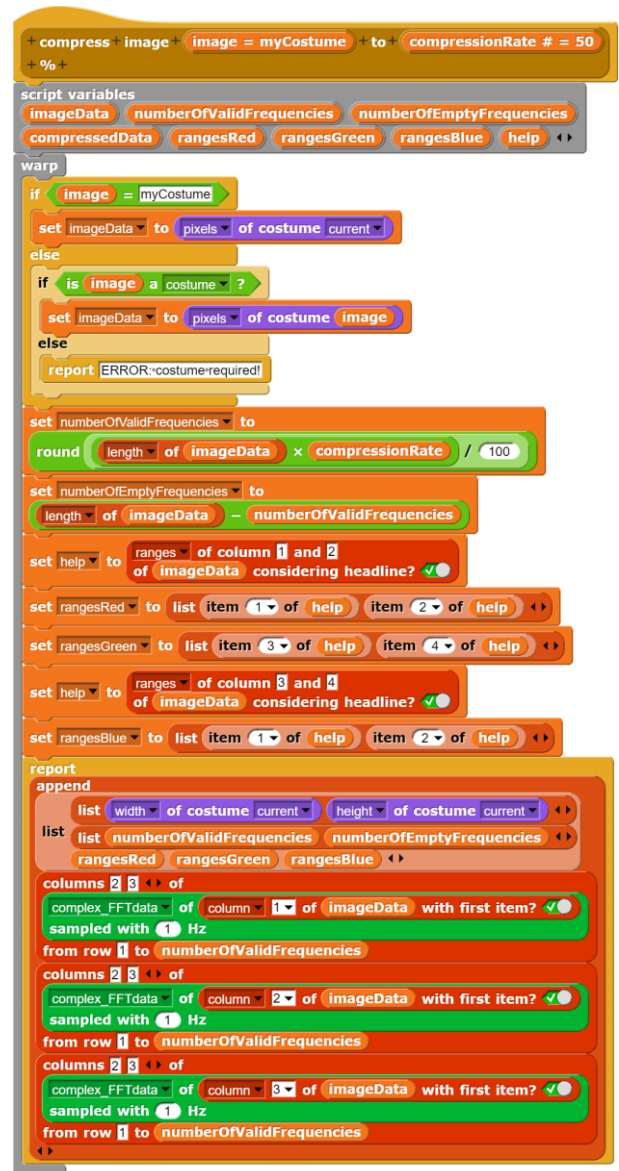
6.15 A simple image compression method with FFT

In the last section we saw that images processed with FFT are actually still quite recognizable, even though a large part of the higher frequencies has been "cut off". We can take advantage of this to reduce the storage space required by images without losing too much of the image quality. However, it is still a lossy compression technique. Parts of JPEG²⁰ compression work similarly.

Our method shall work as follows: We specify a compression rate *compressionRate* and the costume of a sprite as parameter *image*. Then we determine the pixels of the image as *imageData*. Now we can calculate how many frequencies should be "valid" at the desired compression. We call the result *numberOfValidFrequencies*. The remaining *numberOfEmptyFrequencies* are later set to zero and used to "fill up" the FFT result to the correct length. To halfway preserve the contrast range of the original image, we also determine the value ranges of the three color channels *rangesRed*, *rangesGreen* and *rangesBlue*. Together with the *width* and *height* of the image, we store these data, from which the image is later reconstructed, in a list. We add the FFTs of the three colors to this list, but we only store the "valid" number of them. The compressed image thus has the format:

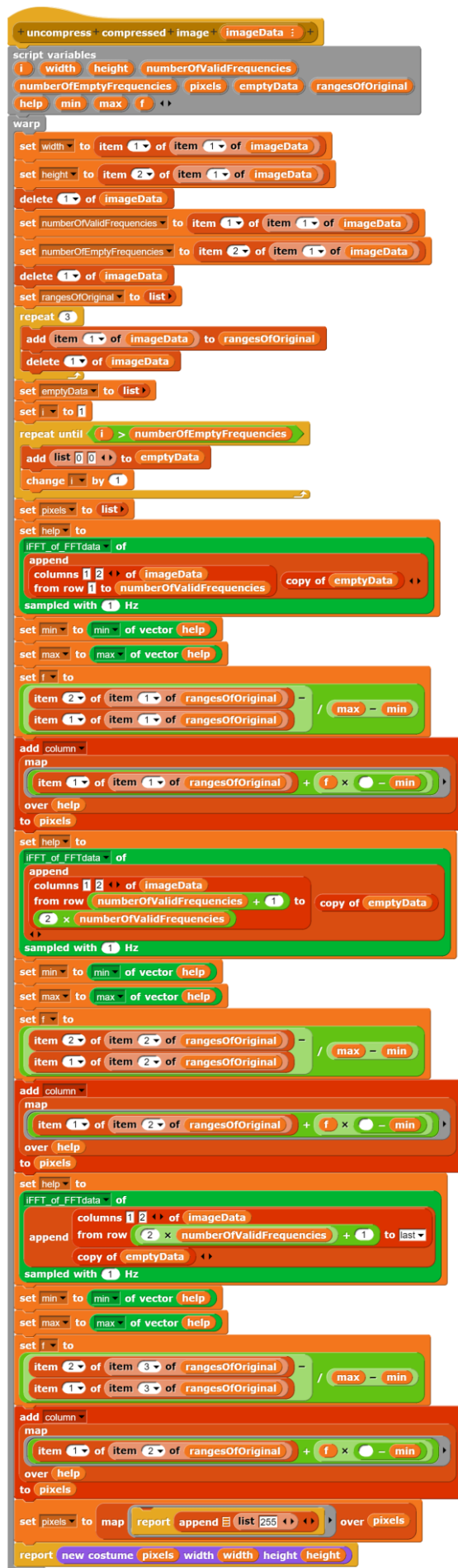
width	height
numberOfValidFrequencies	numberOfEmptyFrequencies
min red	max red
min green	max green
min blue	max blue
data of the three color channels as complex numbers	

We can assign this data to a variable and export its content, for example, or we can write directly to a file.



When decompressing the image, we have to read the stored data and fill the three color channels with the complex number zero. After that, they are transformed back into real number sequences with *iFFT* and assembled into pixels. We afford ourselves some additional luxury: since we have stored the original value ranges of the color channels, we stretch the colors, which are also reduced by the reduced number of frequencies in their value range, back to the original range.

²⁰ <https://de.wikipedia.org/wiki/JPEG>



read out and delete image dimensions

read number of valid or deleted frequencies
and delete

read out and delete original color ranges

create a list with complex zeros of the correct length

IFFT for the red channel

determine current limits of the red color channel

expand the red channel to the original values and insert it into
the pixel list

IFFT for the green channel

determine current limits of the green color channel

expand the green channel to the original values and insert it
into the pixel list

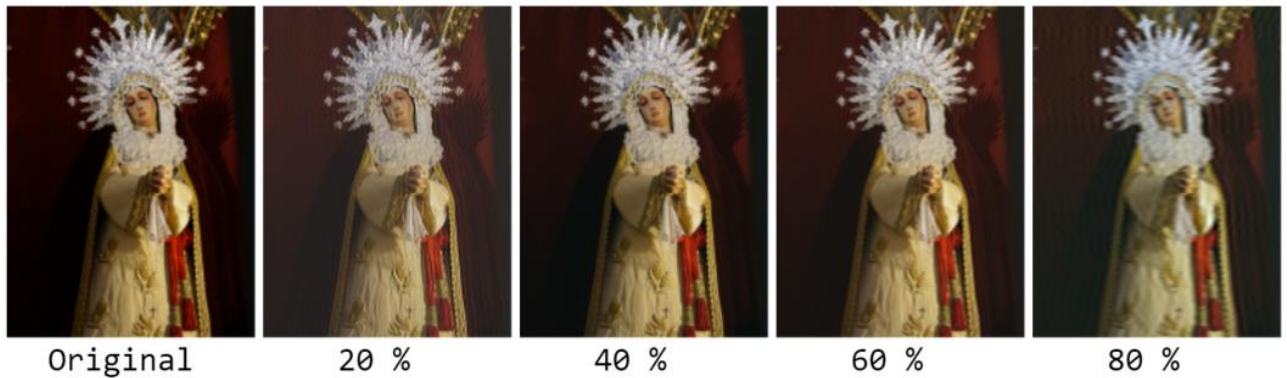
IFFT for the blue channel

determine current limits of the blue color channel

expand the blue channel to the original values and insert it
into the pixel list

insert transparency values and export the costume

For the test we give *Hilberto* a picture as a costume and compress it increasingly.



There we go! 😊

Tasks:

1. Experiment with different image types (portrait, landscape, ...) and compression ratios.
2. Learn about compression methods for images and other data.
3. Invent better compression methods for images, e.g. by not simply truncating frequency ranges, but by limiting yourself to the "essential" frequencies, ...

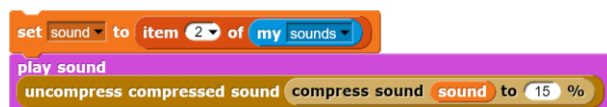
6.16 A simple sound compression method with FFT

Since all data must be in digital form, it doesn't really matter what kind of data is processed with the FFT. So we can compress sound data just as well. It is even easier if we limit ourselves to one channel. Again, this is a lossy compression method. Parts of the MP3 compression²¹ work similarly.

Our method should work as follows: We specify a compression rate *compressionRate* and a sound as parameter *sound*. Then we determine the samples of the image as *samples*. If it should be a sound with several channels, then we take from these only the first one. Now we can calculate how many frequencies should be "valid" for the desired compression. We call the result *numberOfValidFrequencies*. The remaining *numberOfEmptyFrequencies* are later set to zero and used to "fill up" the FFT result to the correct length. The compressed sound thus has the format:

sample rate
numberOfValidFrequencies
numberOfEmptyFrequencies
sound data as complex numbers

When decompressing the sound, we have to read the stored data and fill up the sound data with the complex number zero. After that they are transformed back to real number series with iFFT and returned as reconstructed sound.



Up to a compression of about 15% I hardly hear any differences. But maybe you are younger. 😊



Tasks:

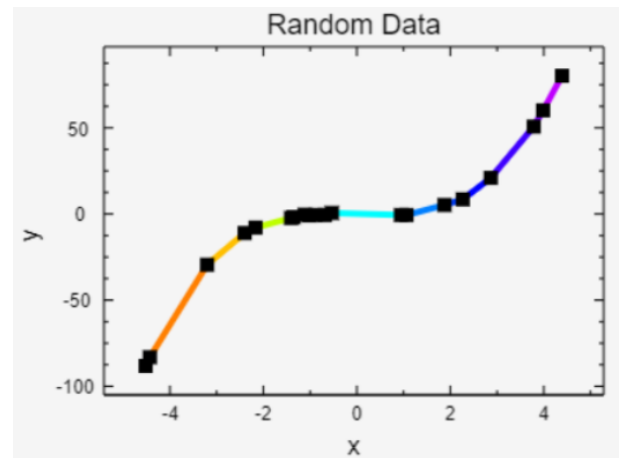
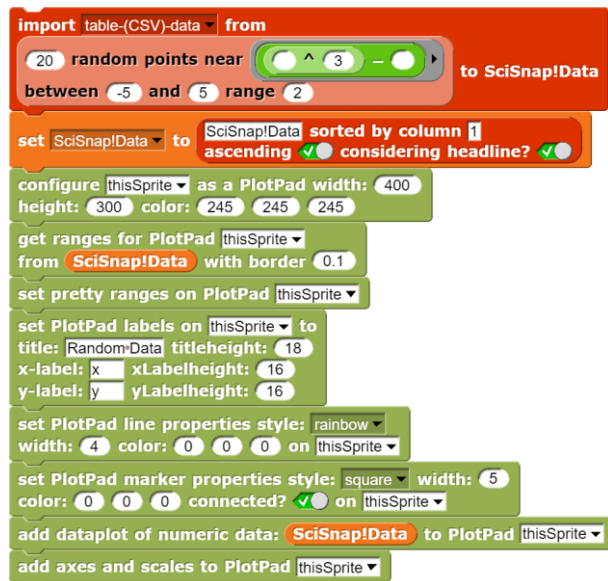
1. Record music, speech, other noises as sounds and convert them to WAV files. Import the sounds to *SciSnap!*.
2. Experiment with the compression method. Have the results evaluated by different people.
3. Learn about the MP3 (and other) compression method for sounds.

²¹ <https://de.wikipedia.org/wiki/MP3>

7 Data related Examples

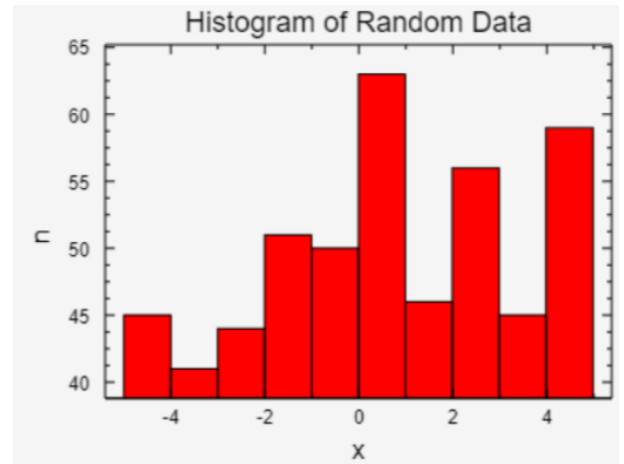
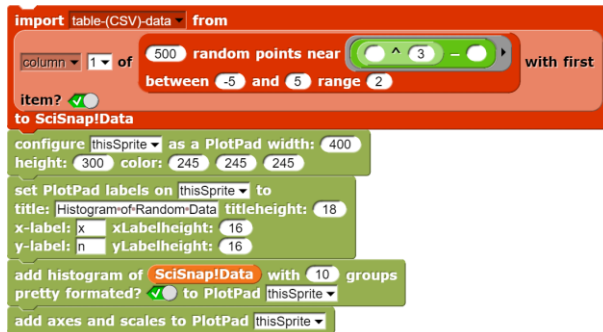
7.1 Data plot of points scattering around a function graph

We create some data points close to the graph to $f(x) = x^3 - x$ and plot them on a *PlotPad*. To make them appear nicely ordered, we sort the points before creating the plot, and because it's Sunday, we connect them in rainbow colors.



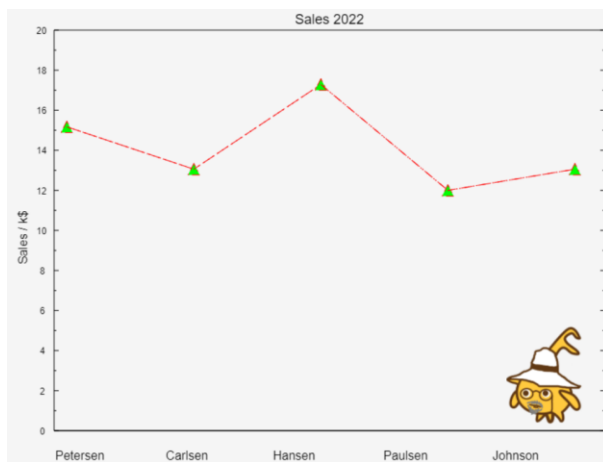
7.2 Histogram of random values

We again create random points scattering around the graph to $f(x) = x^3 - x$, but this time we choose only the first column as the data set. From these values we let create a histogram with 10 columns.



7.3 Plot of mixed data

Text data are often combined with numerical data. An example would be the sales data of different representatives in one year in an area. If we want to plot these graphically, then, for example, the x-axis must be labeled with text data, while the y-axis is treated as before. To create the graph we use the add dataplot of mixed data block of the *PlotPad*. In this case, the *stage* is to serve as the *PlotPad*.



```

set data to
list Petersen 15 list Carlsen 13 list Hansen 17 list Paulsen 12
list Johnson 13

configure theStage as a PlotPad width: 400
height: 300 color: 245 245 245

set PlotPad labels on theStage to
title: Sales-2022 titleheight: 18
x-label: get label from text-data data at column 1 xLabelheight: 16
max. textwidth 8 column spacing 18
y-label: Sales/k$ yLabelheight: 16

set PlotPad line properties style: dash-dot
width: 1 color: 255 0 0 on theStage

set PlotPad marker properties style: triangle width: 15
color: 0 255 0 connected? checked on theStage

set PlotPad scale properties precision: 3 0
textheight: 12 12 number of intervals: 5 10
on theStage

set PlotPad ranges for x: 0 5 y: 0 20
with border? checked of 0.1 pretty formatted? checked
on theStage

add dataplot of mixed data: data
y-scale? checked x-scale? checked to PlotPad theStage

add axes and scales to PlotPad theStage
  
```

7.46 NY CitiBike Tripdata 1: Correlations

As an example of how to use the blocks, let's "dig" a bit into a larger, freely available dataset: the New York CitiBike rental data (NY citibike tripdata: <https://www.citibikenyc.com/system-data>).

We load the data, extract it, and get CSV data of rather big a size, which we load "in one swoop" into the *SciSnap!* data area, the *SciSnap!Data* variable. We get almost 600,000 records.

import table-(CSV)-data
filepicker to SciSnap!Data

	A	B	C	D	E	F	G	H	I	J
1	tripduration	starttime	stoptime	start station	istart station	istart station	istart station	lend station	iend station	rend station
2	695	2013-06-01 (2013-06-01)	444	Broadway & 40.7423543	-73.9891507	434	9 Ave & W 140.7431744	-74.0		
3	693	2013-06-01 (2013-06-01)	444	Broadway & 40.7423543	-73.9891507	434	9 Ave & W 140.7431744	-74.0		
4	2059	2013-06-01 (2013-06-01)	406	Hicks St & M40.6951284	-73.9959506	406	Hicks St & M40.6951284	-73.9		
5	123	2013-06-01 (2013-06-01)	475	E 15 St & Irv40.7352427	-73.9875856	262	Washington	40.6917823	-73.9	
6	1521	2013-06-01 (2013-06-01)	2008	Little West 540.7056925	-74.0167768	310	State St & S40.6892694	-73.9		
7	2028	2013-06-01 (2013-06-01)	485	W 37 St & 540.7503800	-73.9833898	406	Hicks St & M40.6951284	-73.9		
8	2057	2013-06-01 (2013-06-01)	285	Broadway & 40.7345456	-73.9907414	532	S 5 Pl & S 540.710451	-73.9		
9	369	2013-06-01 (2013-06-01)	509	9 Ave & W 240.7454973	-74.0019713	521	8 Ave & W 340.7509673	-73.9		
10	1829	2013-06-01 (2013-06-01)	265	Stanton St & 40.7222934	-73.9914753	436	Hancock St & 40.6821656	-73.9		
11	829	2013-06-01 (2013-06-01)	404	9 Ave & W 140.7405826	-74.0055096	303	Mercer St & 40.7236273	-73.9		
12	1316	2013-06-01 (2013-06-01)	423	W 54 St & 940.7658494	-73.9869050	314	Cadman Plaz	40.69383	-73.9	
13	1456	2013-06-01 (2013-06-01)	502	Henry St & C40.714215	-73.981346	532	S 5 Pl & S 540.710451	-73.9		
14	386	2013-06-01 (2013-06-01)	241	DeKalb Ave	40.6898103	-73.9749312	365	Fulton St & 40.6822316	-73.9	
15	924	2013-06-01 (2013-06-01)	486	Broadway & 40.7462009	-73.9885572	521	8 Ave & W 340.7509673	-73.9		
16	1233	2013-06-01 (2013-06-01)	527	E 33 St & 240.744023	-73.9760056	296	Division St & 40.7141308	-73.9		
17	512	2013-06-01 (2013-06-01)	309	Murray St & 40.7149787	-74.013012	300	Shevchenko	40.728145	-73.9	

Their column headers we split off from the table.

set headlines to row 1 of SciSnap!Data with first item? ✓

These data can be analyzed in very different ways. We will do this later, but for now we will limit ourselves to the question of whether there is a correlation between gender and loan duration. Obviously, we only need columns 1 and 15 for this, so we delete the other columns.

set SciSnap!Data to columns tripduration gender of SciSnap!Data from row 1 to last

First of all, let's look at the mean values for the different genders (0: unknown, 1 male, 2: female):

set result to mean of column A of SciSnap!Data grouped by column B considering headline? ✓

	A	B
1	tripduration	gender
2	695	1
3	693	1
4	2059	0
5	123	1
6	1521	1
7	2028	0
8	2057	1
9	369	1
10	1829	1
11	829	1
12	1316	1

	value	mean
1	0	1753.2988186
2	1	1063.5487225
3	2	1233.2494452

headlines
1 tripduration
2 starttime
3 stoptime
4 start station id
5 start station name
6 start station latitude
7 start station longitude
8 end station id
9 end station name
10 end station latitude
11 end station longitude
12 bikeid
13 usertype
14 birth year
15 gender

That's what we thought! 😊

And what about the correlation? First, we delete the data with the "unknown" gender. There are still about 340,000 data records left. For these, we calculate the correlation coefficient between column 1 and 2 - and get the result shown opposite.

set SciSnap!Data to select rows of SciSnap!Data where column gender is different-from 0

set result to correlation of column tripduration and gender of SciSnap!Data considering headline? ✗

result 0.014741637

And what does this number want to tell us now??? We don't know - but we can read up and learn! 😊

7.5 New York Citibike Tripdata 2: usage of bikes

Let's take a look at who actually rides a bike in New York. To do this, we download the rental data from NY Citibike for a month onto the computer, which are the almost 600,000 data records already mentioned. Let's take a closer look.

import table-(CSV)-data from
filepicker to SciSnap!Data

Table view																
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
577704	tripduration	starttime	stoptime	start station	start station	start station	start station	end station	end station	end station	end station	bikeid	usertype	birth year	gender	
1	695	2013-06-01 (2013-06-01)	444	Broadway & 40.7423543	-73.9891507	434	9 Ave & W 140.7431744	-74.0036644	19678	Subscriber	1983	1				
2	693	2013-06-01 (2013-06-01)	444	Broadway & 40.7423543	-73.9891507	434	9 Ave & W 140.7431744	-74.0036644	16649	Subscriber	1984	1				
3	2059	2013-06-01 (2013-06-01)	406	Hicks St & M40.6951284	-73.9959506	406	Hicks St & M40.6951284	-73.9959506	19599	Customer	NULL	0				
4	123	2013-06-01 (2013-06-01)	475	E 15 St & Irv40.7352427	-73.9875856	262	Washington 40.6917823	-73.9737299	16352	Subscriber	1960	1				
5	1521	2013-06-01 (2013-06-01)	2008	Little West S40.7056925	-74.0167768	310	State St & S40.6892694	-73.9891286	15567	Subscriber	1983	1				
6	2028	2013-06-01 (2013-06-01)	485	W 37 St & 540.7503800	-73.9833898	406	Hicks St & M40.6951284	-73.9959506	18445	Customer	NULL	0				
7	2057	2013-06-01 (2013-06-01)	285	Broadway & 40.7345456	-73.9907414	532	S 5 Pl & S 5 40.710451	-73.960876	15693	Subscriber	1991	1				

Of course, we still need to find out from the source what the data actually means - that is, look at the metadata. For the gender we learn that 0: *unknown*, 1: *male* and 2: *female*. For the columns "tripduration" and "gender" we determine some data:

result		
	A	B
4	value	mean
1	0	1753.2988186817752
2	1	1063.5487225418608
3	2	1233.249445298994

We already know the average duration of borrowing, based on gender, from the last example.

Let's see if they are lazier on Broadway:

result 1380.553088413

set result to	mean of vector
column A of	select rows of SciSnap!Data where column start:stationid is equal to 444 with first item? <input checked="" type="checkbox"/>

I see. Probably Central Park is even worse!

result 2230.303350254

set result to	mean of vector
column A of	select rows of SciSnap!Data where column start:stationid is equal to 2006 with first item? <input checked="" type="checkbox"/>

All right. All prejudices do not have to be true. 😊

Tasks:

1. But maybe only the women at Central Park ride their bikes more. Check it out.
2. There is not only one rental station at Central Park. Determine appropriate averages for the entire area.
3. Are there actually rental data for other parts of the city? Do a search and compare the results with Manhattan.
4. Determine the average borrowing times per weekday, in total and for individual stations. Are there differences? Why?
5. Above, the mean borrowing time was calculated in relation to gender. You could also do it the other way around. Would that be complete nonsense or are there questions for which that would make sense?

7.6 New York Citibike Tripdata 3: World Map Library

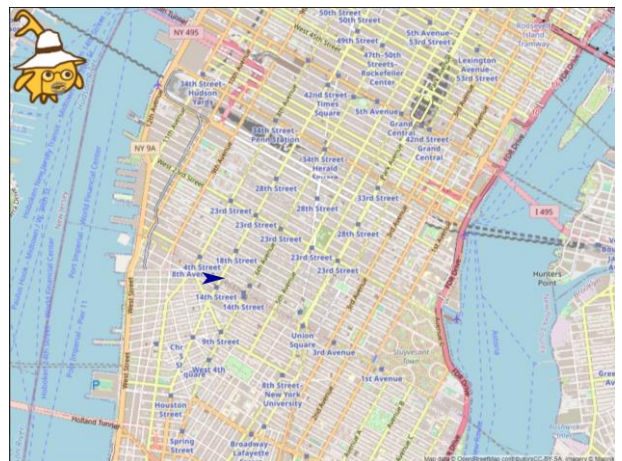
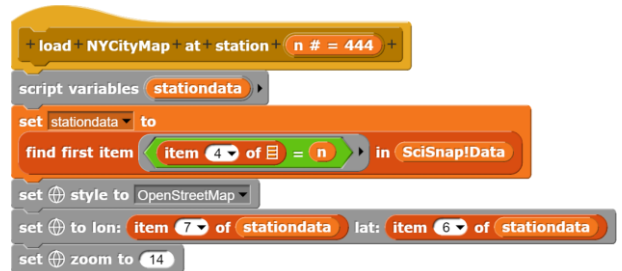
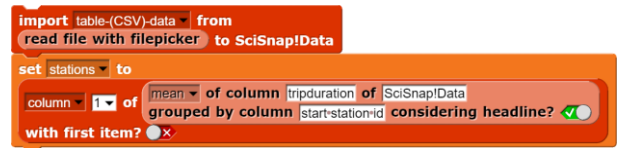
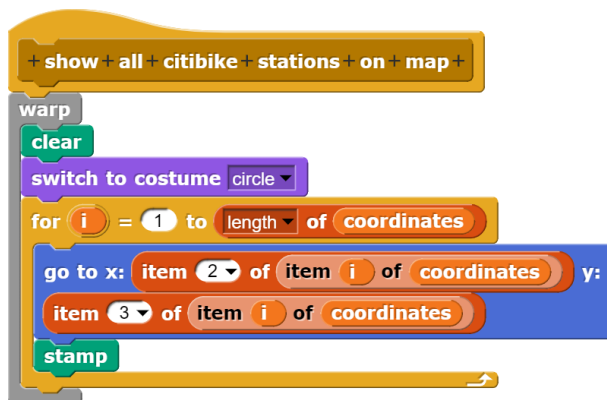
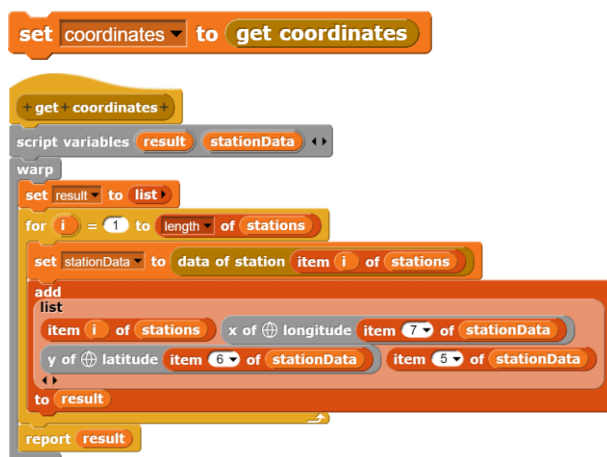
Even in New York, bicycling has become "hip" and the borrowing data can be loaded as CSV files. We do this as in the previous examples with this dataset. Since we also want to create graphics, we configure a sprite as *PlotPad*.

Let's see where you can rent bicycles. For the overview we extract the rental stations from the total list, e.g. by grouping them by the name of the starting station (column 5) and selecting only this column as the result. We still get 337 stations. Since geographical longitude and latitude of the borrowing stations are given, it makes sense to use the *World Map Library* of *Snap!*. We write a small block for this, which displays the surroundings of a borrowing station as a map.

After that, we look for the data of a station ...



... and thus set up the list of coordinates of the stations²².



With this data we can send *Hilberto* to the individual positions and ask him to leave circles there with the stamp block, for example.



At least in midtown Manhattan, I don't think we have to worry about finding a borrowing station!

²² This will take some time!

7.7 New York Citibike Tripdata 4: Lending diagrams

Let's take a closer look at the rental station Broadway - corner 41 Street (No. 465). To do this, we pull all records from the total list that start or end at this station. That's 5054 operations this month. Times are entered in this list along with the date. We can throw this out (*split with " "*) and reduce it to the hour (*split with ":"*). We then have a numerical scale with the unit "hour". Now we can see what is going on at the station in the individual hours of the day.

```
set lendingData to
reduce time columns of
append lendings at station 465 returns at station 465
```

lendingData			
5054	A	B	C
1	627	00	00
2	586	00	00
3	601	00	00
4	993	05	05
5	34966	07	17
6	2314	10	11
7	843	11	11
8	250	12	12
9	707	12	12
10	344	13	13
11	2177	13	13
12	2156	13	13
13	670	13	13
14	2100	14	15
15	1394	15	15
16	820	16	16
17	13649	16	20

From this we build a diagram.

```
configure thisSprite as a PlotPad width: 500
height: 400 color: 245 245 245
set PlotPad labels on thisSprite to
title: join Activities at item 5 of item 1 of lendingData
titleheight: 18
x-label: hour xLabelheight: 16
y-label: number of activities yLabelheight: 16
set PlotPad ranges for x: 0 24 y: 0
max of vector column 2 of plotdata with first item?
with border? of 0.1 pretty formatted?
on thisSprite
set PlotPad marker properties style: o circle width: 5
color: 0 255 0 connected? on thisSprite
add dataplot of numeric data: plotdata to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```

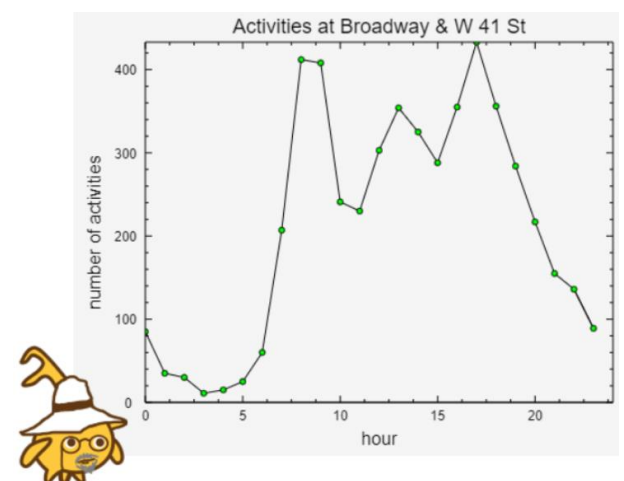
```
+ selection = lendings + at station + n # = 444 +
if selection = lendings
report keep items item 4 of = n from SciSnap!Data
else
report keep items item 8 of = n from SciSnap!Data
```

lendings
station 444
lendings
returns

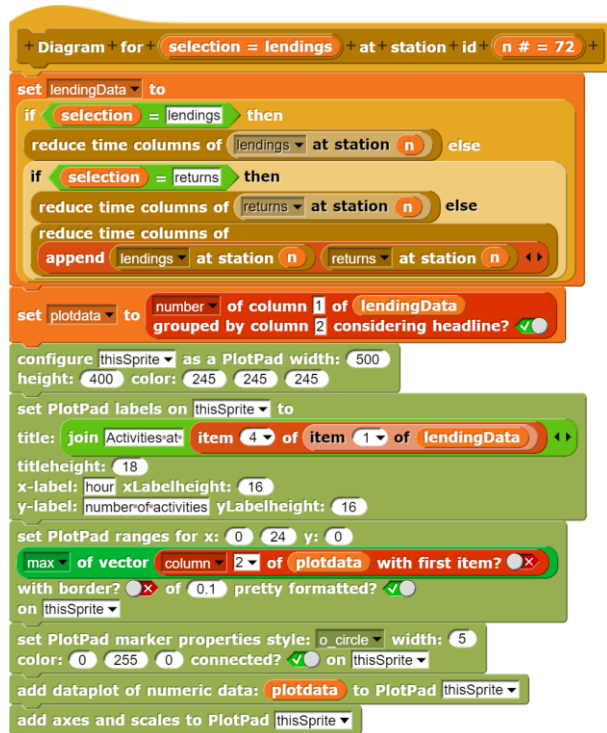
```
+ reduce time columns of + data : +
script variables result
warp
set result to list
add column column 1 of data with first item? to result
add column
map item 1 of split item 2 of split by by over to
column 2 of data with first item?
result
add column
map item 1 of split item 2 of split by by over to
column 3 of data with first item?
result
add column column 5 of data with first item? to result
report result
```

And we can represent this graphically as usual. We simply count how many there were in each hour of the day.

```
set plotdata to number of column 1 of lendingData
grouped by column 2 considering headline?
```



Of course, we can combine that into one block, still leaving open the decision of whether we want to record borrowings, returns, or both.



lendings

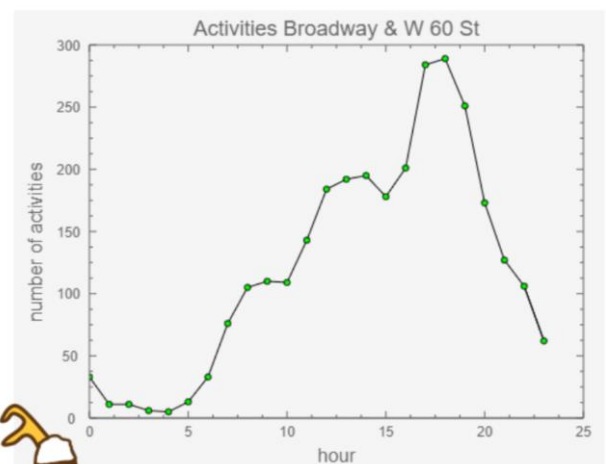
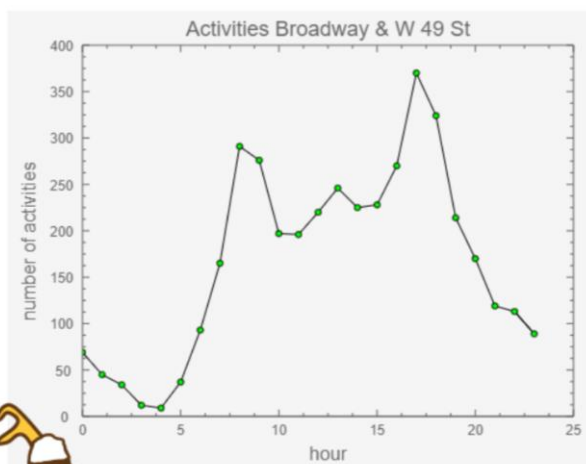
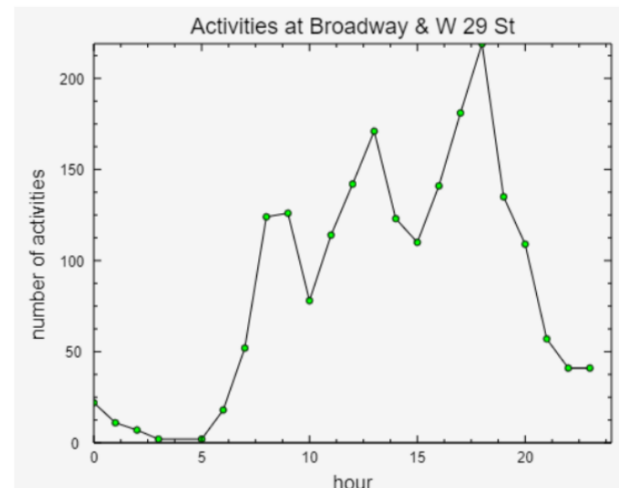
returns

both

aggregate data

create diagram

A few streets away, it looks very similar. Is this a general pattern?



Well, at Central Park people get up later and the tourists are not there yet. But the museums always close at the same time.

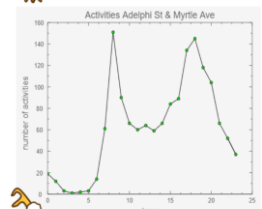
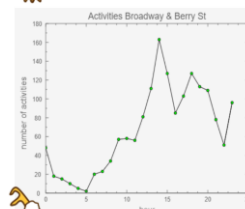
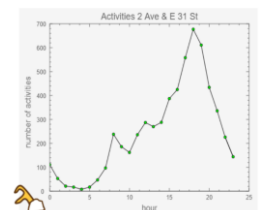
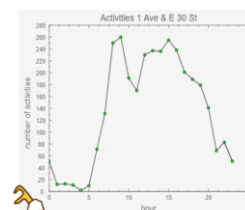
So, what can our programs learn from this data?

- For example, we could predict from the usual departures and arrivals as well as the actual stock whether enough bikes will be returned to a station in time or whether it would be better to transport some there.
- We could determine from mean path lengths which batteries are being used for eBikes.
- We could determine whether women or men are more likely to borrow the bikes at a particular time of day and then make sure the supply is right. We could do the same for the age of the borrowers.
- We could determine the borrowing data per bike and predict when repairs will be due. We could also do this as a function of the location of the stations, for example.
- We could try to generalize distributions from some stations so that predictions for others can be derived from them. So, when museums close at Central Park, the program can "learn" from the old data in which districts bikes are likely to be dropped off and when and warn if there are not enough free slots there.

etc.

Tasks:

1. Break down the stations' activities according to arrivals and departures.
2. Write a forecasting function that warns when there is a threat of a bike shortage at a station in the next few hours.
3. For specific stations, graphically represent the connections to the most selected drop-off stations on the map by direct lines. Choose line thickness according to the number of borrows and colors according to the station. Do clusters form?
4. With the help of correlations, determine whether there are correlations in the borrowing behavior (e.g. with regard to the times of day, the location, ...) with the gender, the age, the status of the borrowers. You may need to replace the data with numerical data beforehand - similar to the times. Discuss possible consequences.
5. For a small section of Midtown (where everything is nice and rectangular), determine the coordinates of the street corners. Then develop a router that shows the shortest path to the nearest Citibike station.
6. Borrowing rates depending on the time of day show some differences in different areas of Manhattan. Examine similarities and differences systematically and try to explain the results.



7.8 Income data from the US Census Income Dataset (Quelle: [Census])

We want to dig into some data and therefore download the *Census Income Dataset* from the net.²³ The corresponding CSV file can be loaded from the file directory in *SciSnap!Data* and displayed immediately. It contains 32562 records. Double-clicking or right-clicking on it and selecting "open in dia-

import table-(CSV)-data from
filepicker to SciSnap!Data

32562	A	B	C	D	E	F	G	H	I	J	K	L	M
1	age	workclass	final weight	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-w
2	39	State-gov	77516	Bachelors	13	Never-marri	Adm-clerica	Not-in-famil	White	Male	2174	0	40
3	50	Self-emp-ne	83311	Bachelors	13	Married-civ	Exec-mana	Husband	White	Male	0	0	13
4	38	Private	215646	HS-grad	9	Divorced	Handlers-cl	Not-in-famil	White	Male	0	0	40
5	53	Private	234721	11th	7	Married-civ	Handlers-cl	Husband	Black	Male	0	0	40
6	28	Private	338409	Bachelors	13	Married-civ	Prof-special	Wife	Black	Female	0	0	40
7	37	Private	284582	Masters	14	Married-civ	Exec-mana	Wife	White	Female	0	0	40
8	49	Private	160187	9th	5	Married-spc	Other-servic	Not-in-famil	Black	Female	0	0	16
9	52	Self-emp-ne	209642	HS-grad	9	Married-civ	Exec-mana	Husband	White	Male	0	0	45
10	31	Private	45781	Masters	14	Never-marri	Prof-special	Not-in-famil	White	Female	14084	0	50
11	42	Private	159449	Bachelors	13	Married-civ	Exec-mana	Husband	White	Male	5178	0	40
12	37	Private	280464	Some-colle	10	Married-civ	Exec-mana	Husband	Black	Male	0	0	80

Which correlations could be found in this?

Our data blocks don't help that much at first, because they mostly process numeric data. If we want to use them, we have to scale the columns in such a way that numerical contents result. In the simplest case, we replace texts with numerical values - and should think carefully about what consequences this might have with regard to their interpretation.

Let's start with the last column: Income values are given for only two ranges: less than or greater than \$50,000. We assign values 1 and 2 to these ranges. (Or 0 and 1, or -1 and +1, or 0 and 100, or ...



Would these changes have consequences? To avoid changing the original data, we create a variable *income* and store the changed values there by copying column 15 (*income*) without the first value (the *heading*) into this variable and then using the *map...over...* block to change the contents. Anyway, that's what we try to do. Unfortunately, when we look at the result again as a table, we only get the unchanged column 13.

32561	items
1801	<=50K
1802	<=50K
1803	<=50K
1804	>50K
1805	<=50K
1806	<=50K
1807	<=50K
1808	<=50K
1809	<=50K
1810	<=50K
1811	<=50K

What's going on? We look at the first element of *income* and check if it is a string. It is, but it is longer than we thought:



If we split them at the spaces, we see what's going on: leading spaces have crept in. Those crooks!



²³ This is one of the training datasets for machine learning.

So we have to throw out the leading blanks before. This works now: our variable *income* now only contains the values 1 and 2, as we can quickly check by looking at it.

```

set income to column income of SciSnap!Data with first item?
set income to
map report item 2 of split by over income
set income to
map report if <= 50K then 1 else if > 50K then 2 else over income

```

income	items
32561	
2841	2
2842	1
2843	1
2844	2
2845	2
2846	2
2847	1
2848	1

What does this income now depend on?

Maybe from age? We combine column 1 (age) and our modified income column into a new table called *testdata*.

```

set testdata to empty table
add column column age of SciSnap!Data with first item? to testdata
add column income to testdata

```

We describe the relationship between age and income by the correlation coefficient. The calculation is simple:

```

set correlation-coefficient to correlation of column 1 and 2 of testdata considering headline?

```

And what does that mean?

correlation coefficient 0.234037103

testdata	A	B
32561		
1	39	1
2	50	1
3	38	1
4	53	1
5	28	1
6	37	1

Tasks:

1. Find out the meaning of the correlation coefficient and interpretation of the obtained value. What does the value "0.2340..." mean?
2. In this case, does the correlation coefficient depend on the type of numerical scaling of the data (1 and 2, -1 and 1, ...)? Check.
3. Determine other correlation coefficients, e.g. between education and income, country of origin and income, marital status and income, country of origin and occupation, ...
4. Find out if and when the scaling of non-numerical data can have an influence on the result.

7.9 Covid-19 data analysis

We upload the Johns Hopkins University Covid-19 data from 2020/3/1 to 2020/4/19 for four counties to the data section of *SciSnap!* and get:

Since we are only interested in the pure data, we pick out the relevant data range for a country.

set data to subsection of table-data in SciSnap!Data from B 5 to C 55

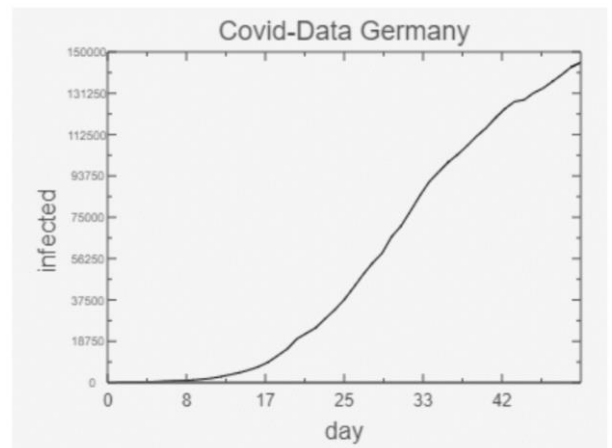
data		
51	A	B
1	Day	Germany
2	1	119
3	2	152
4	3	190
5	4	264
6	5	402

55	A	B	C	D	E	F
1		Covid-10 Infections from 1.3.2020				
2		origin: Johns-Hopkins-University				
3						
4			Infected			
5	Date	Day	Germany	Italy	USA	China
6	1.3.	1	119	1694	43	79826
7	2.3.	2	152	2036	67	80026
8	3.3.	3	190	2502	88	80151
9	4.3.	4	264	3089	117	80271
10	5.3.	5	402	3858	186	80422
11	6.3.	6	641	4636	232	80573
12	7.3.	7	797	5883	351	80652
13	8.3.	8	900	7375	469	80699
14	9.3.	9	1146	9172	535	80735
15	10.3.	10	1567	10149	892	80757
16	11.3.	11	1968	12462	1214	80785
17	12.3.	12	2747	12462	1596	80793
18	13.3.	13	3677	17660	1647	80801
19	14.3.	14	4587	21157	2656	80827
20	15.3.	15	5815	24747	3338	80848

Let's first get an overview of this:

```

configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on thisSprite to
title: Covid-Data-Germany titleheight: 18
x-label: day xLabelheight: 16
y-label: infected yLabelheight: 16
set PlotPad ranges for x: 0 50 y: 0 150000
with border? of 0.1 pretty formatted?
on thisSprite
set PlotPad line properties style: continuous
width: 1 color: 0 0 0 on thisSprite
set PlotPad marker properties style: none width: 5
color: 0 0 0 connected? on thisSprite
set PlotPad scale properties precision: 0 0
textheight: 12 8 number of intervals: 10 8
on thisSprite
add dataplot of numeric data: data to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
  
```



Then let's try it with a semi-logarithmic representation ...

```

add column
append
list in(data) in of column B of data with first item? to
data
  
```

... pick out the two interesting columns ...

```

set data to columns Day ln(data) of data
from row 2 to last
  
```

... and fit the graph: We show the semi-logarithmically plotted data and the regression lines for the two halves of the data.

```

configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245

set PlotPad costume properties width: 400 height: 300
back color: 245 245 245 front color: 80 80 80
offsets: 0 0 on thisSprite

set PlotPad labels on thisSprite to
title: Covid-Data-Germany-semi-logarithmic titleheight: 18
x-label: day xLabelheight: 16
y-label: ln(infected) yLabelheight: 16

set PlotPad ranges for x: 0 50 y: 0 15
with border? ☒ of 0.1 pretty formatted? ☒
on thisSprite

set PlotPad line properties style: continuous
width: 1 color: 0 0 0 on thisSprite

set PlotPad marker properties style: none width: 5
color: 0 0 0 connected? ☒ on thisSprite

set PlotPad scale properties precision: 0 0
textheight: 12 8 number of intervals: 10 8
on thisSprite

add dataplot of numeric data: select rows of data where
column A is less-than 51 to PlotPad
thisSprite

set PlotPad line properties style: continuous
width: 1 color: 255 0 0 on thisSprite

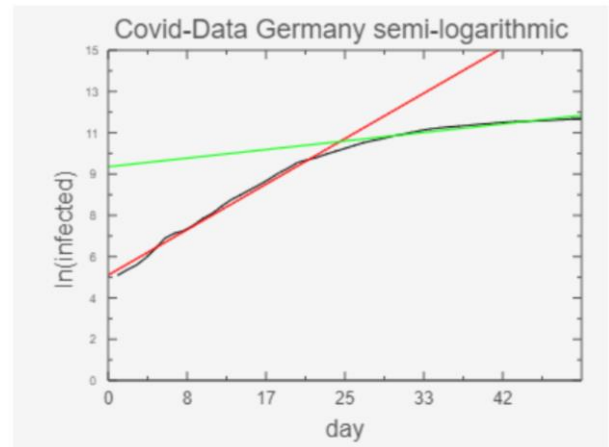
add graph
regression line parameters of subsection of table-data in data from
A 1 to B 25 to
PlotPad thisSprite

set PlotPad line properties style: continuous
width: 1 color: 0 255 0 on thisSprite

add graph
regression line parameters of subsection of table-data in data from
A 26 to B 50 to
PlotPad thisSprite

add axes and scales to PlotPad thisSprite

```



That's when hope arose!



Tasks:

1. Plot the data for the other countries as well.
2. Try to determine whether there are correlations between the data series.

7.10 Star spectra [UniGOE]

Stars shine in different colors because they have different temperatures. In addition, the spectra differ in their absorption lines. We want to investigate this a little more closely.

We get some star spectra (source: [UniGOE]) and save them as a text file. We read such a file and split it into a list *data*. The first line contains the star name after the column labels. We isolate it and save it as *starname*.

We now know what the star is called. If you search for it on the Internet, you will find a lot of information about it. So that we can repeat the loading process with other data, we encapsulate it in its own block. After its execution, the actual star data are available as a lightly prepared table. Unpleasant about it is the strongly different magnitude of the data in the two columns. We therefore normalize the second column using the mean value and store the result as *normalizedData*.

data				normalizedData		
2799	A	B	C	2799	A	B
1	351.00	8.1860e-13	0.3586	1	351.00	0.896537451
2	351.14	8.1770e-13	0.3584	2	351.14	0.895551764
3	351.28	8.3890e-13	0.3680	3	351.28	0.918770176
4	351.42	8.4400e-13	0.3704	4	351.42	0.924355740
5	351.56	8.3100e-13	0.3649	5	351.56	0.910118033
6	351.70	8.3270e-13	0.3659	6	351.70	0.911979887
7	351.84	8.3740e-13	0.3682	7	351.84	0.917127366
8	351.98	8.3200e-13	0.3661	8	351.98	0.911213241
9	352.12	8.0760e-13	0.3555	9	352.12	0.884490161
10	352.26	7.8450e-13	0.3456	10	352.26	0.859190851
11	352.40	7.9900e-13	0.3363	11	352.40	0.835534351
12	352.54	7.8450e-13	0.3354	12	352.54	0.832706231

The normalized data can be used to quickly create a plot on a *PlotPad*.

```

+ show + spectrum +
configure PlotPad as a PlotPad width: 500
height: 400 color: 245 245 245
set PlotPad labels on PlotPad to
title: join Spectrum of: starname titleheight: 18
x-label: wavelength/nm xLabelheight: 16
y-label: normalized-flux yLabelheight: 16
set PlotPad ranges for x: 300 800 y: 0 3
with border? of 0.1 pretty formatted? on PlotPad
set PlotPad marker properties style: none width: 5
color: 0 0 0 connected? on PlotPad
add dataplot of numeric data: normalizedData to PlotPad PlotPad
add axes and scales to PlotPad PlotPad

```

set data to read file with filepicker

starname HD 116608

```

# nm Flux(10mW/m2/nm) for star HD 116608
351.00 8.1860e-13 0.3586
351.14 8.1770e-13 0.3584
351.28 8.3890e-13 0.3680
351.42 8.4400e-13 0.3704
351.56 8.3100e-13 0.3649
351.70 8.3270e-13 0.3659
351.84 8.3740e-13 0.3682
351.98 8.3200e-13 0.3661
352.12 8.0760e-13 0.3555
352.26 7.8450e-13 0.3456
352.40 7.6290e-13 0.3363
352.54 7.6040e-13 0.3354
352.68 7.6470e-13 0.3375
352.82 7.9000e-13 0.3489
352.96 8.2580e-13 0.3649
353.10 8.1020e-13 0.3582
353.24 7.8800e-13 0.3486
353.38 8.0680e-13 0.3571
353.52 8...

```

set data to split data by line

set starname to get starname from item 1 of data

+ get + starname + from + text +

script variables result

set result to split text by

if length of text item 8 of result > 1

report join item 7 of result item 8 of result

else

report join item 7 of result item 9 of result

+ load + star + data +

set data to split read file with filepicker by line

set starname to get starname from item 1 of data

delete 1 of data

set data to map split by tab over data

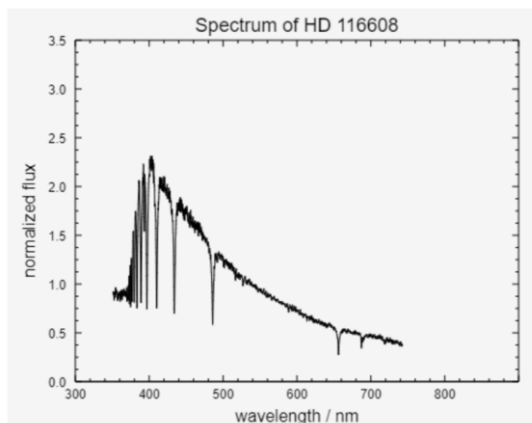
delete last of data

delete column 3 of data

set normalizedData to list

add column column 1 of data with first item? to normalizedData

add column column 2 of data with first item? normalized by mean to normalizedData

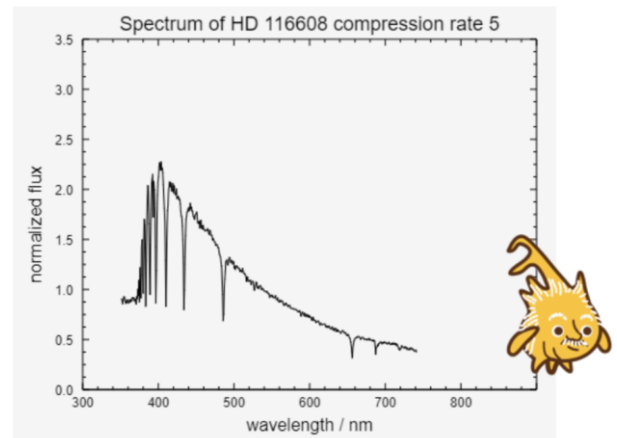


One can see well the sloping course with some striking absorption lines. But do we need all spectral data at all for this insight? Maybe it is enough to reduce the amount of data by averaging. We introduce a compression factor *compressionRate* and complete the script before the diagram generation.

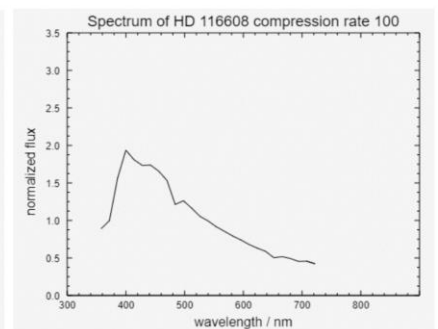
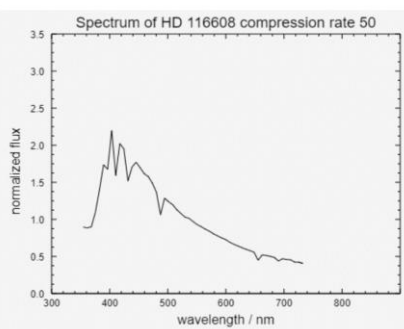
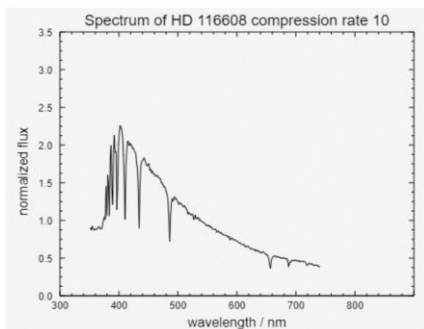
```

+ show + spectrum + with + compression + rate + compressionRate # = 1 +
script variables compressedData
set compressedData to normalizedData compressed with
factor compressionRate by averaging
configure PlotPad as a PlotPad width: 500
height: 400 color: 245 245 245
set PlotPad labels on PlotPad to
title: join Spectrum-of starname compression-rate compressionRate
titleheight: 18
x-label: wavelength/nm xlabelheight: 16
y-label: normalized-flux ylabelheight: 16
set PlotPad ranges for x: 300 800 y: 0 3
with border? of 0.1 pretty formatted?
on PlotPad
set PlotPad marker properties style: none width: 5
color: 0 0 0 connected? on PlotPad
add dataplot of numeric data: compressedData to PlotPad PlotPad
add axes and scales to PlotPad PlotPad

```



The factor 5 does not change much. So let's keep trying.



It can be seen that the temperature-dependent course of the spectrum is hardly changed. Only the absorption lines are lost. Thus, the type of the spectrum should be described by an interpolation polynomial of e.g. 4th degree.

```

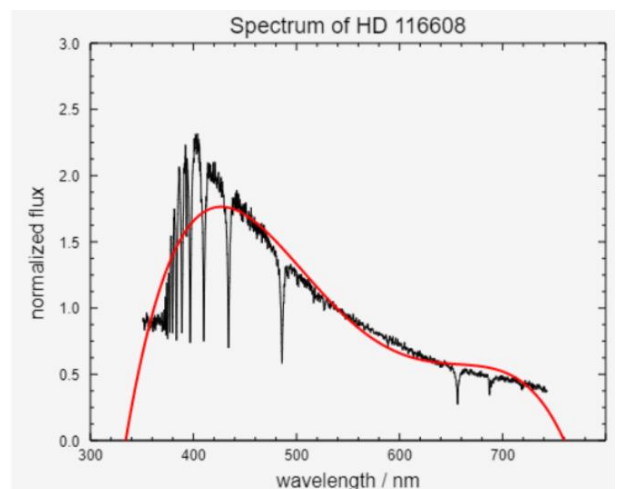
+ interpolation + polynomial + for + data +
script variables polynomialData compressedData
set compressedData to data compressed with
factor 100 by averaging
set polynomialData to list
add item 1 of compressedData to polynomialData
add
item round length of compressedData / 4 of compressedData
to polynomialData
add
item 2 x round length of compressedData / 4 of
compressedData
to polynomialData
add
item 3 x round length of compressedData / 4 of
compressedData
to polynomialData
add item last of compressedData to polynomialData
report polynomial interpolation for points polynomialData

```

```

set PlotPad line properties style: continuous
width: 2 color: 255 0 0 on PlotPad
add graph interpolation polynomial for normalizedData
to PlotPad PlotPad

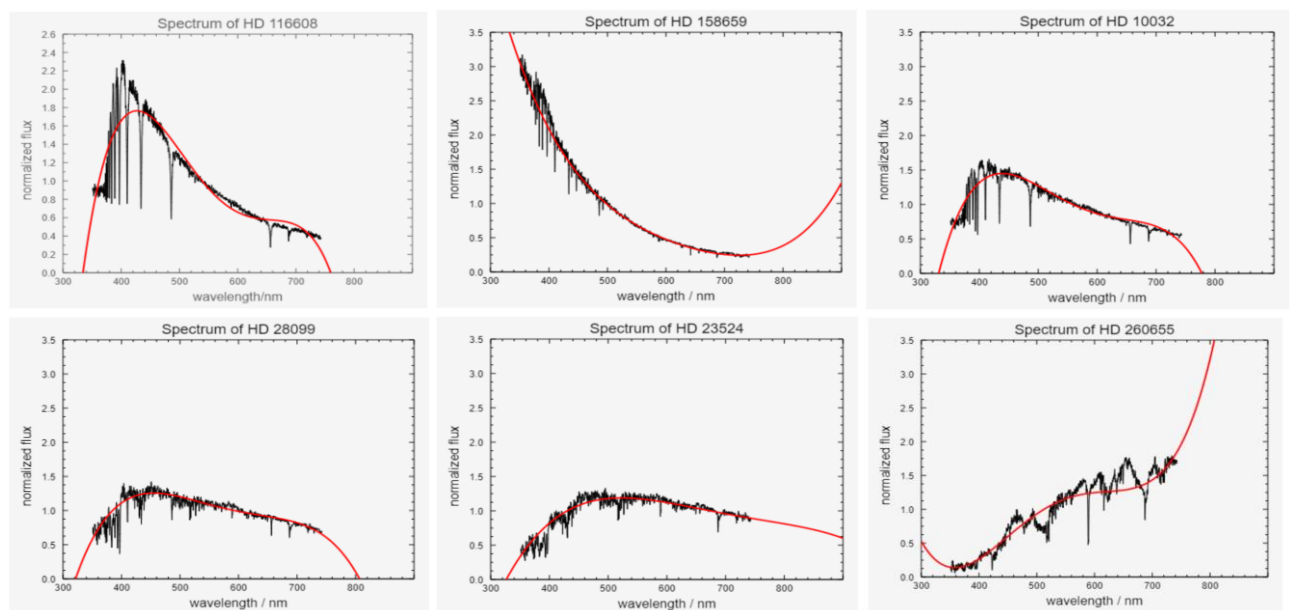
```



So, this works very well! If we log the polynomial parameters at the same time during the examination, then we can easily distinguish the star types on the basis of the parameter ranges.

7	A	B	C	D	E	F
1	star name	a4	a3	a2	a1	a0
2	HD 116608	-1.2493580327340172e-9	0.0000028868621087800814	-0.002459579857425176	0.9103088865090065	0.9103088865090065
3	HD 158659	1.565259017017166e-10	-3.963032080178107e-7	0.0003879661846290463	-0.17622312866994078	-0.17622312866994078
4	HD 10032	-7.27005023271818e-10	0.000001694929847991264	-0.001462425925103779	0.5500801501694278	0.5500801501694278
5	HD 28099	-4.0018935572381893e-10	9.399457129604694e-7	-0.0008209889485783107	0.3141072191721327	0.3141072191721327
6	HD 23524	-8.18301248511472e-11	2.3253458278204257e-7	-0.00024615800544876965	0.11348374829256708	0.11348374829256708
7	HD 260655	6.248027476637483e-10	-0.000001337322548726115	0.0010450333683869723	-0.3486709605339992	-0.3486709605339992

If you feed a neural network with the polynomial coefficients, it quickly learns to roughly assign a diagram to a star type. The program can "learn" on the basis of the old data which parameter intervals belong to which star classes. If one enters the data of a new star, then it determines the coefficients of the polynomial and gives afterwards a well-founded prognosis, around which kind of star it could concern.

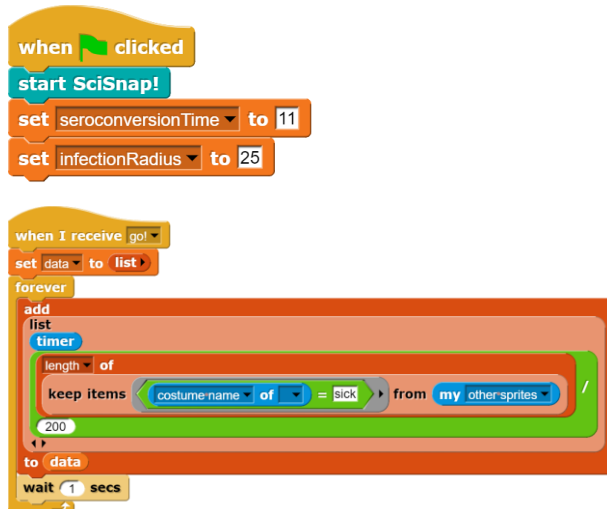


Tasks:

1. Set up an interpolation polynomial of as low a degree as possible for each of the uncompressed spectrum data. Which points should be chosen for this? Are there any differences between these polynomials and the results of the procedure shown above?
2. Develop a script that assigns an unknown spectrum to one of the types that have appeared so far.
3. Develop a procedure to examine the most prominent absorption lines in more detail. Plot them magnified for stars of the same class and try to determine differences "automatically". Discuss your ideas before realization.

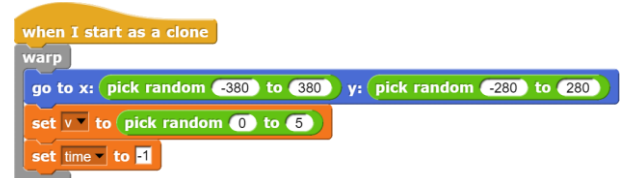
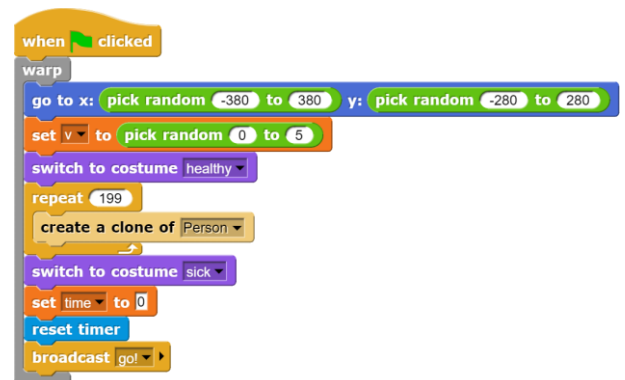
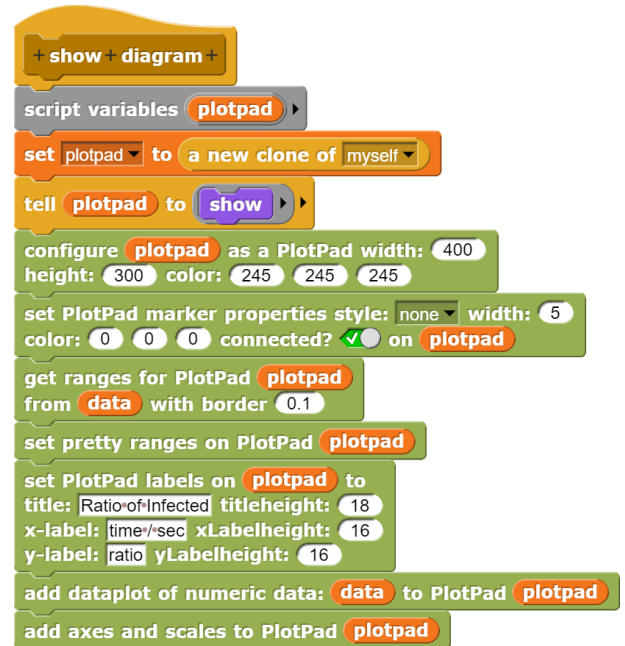
7.11 Flu wave simulation

We want to simulate an influenza wave in the simplest possible way, controlled by only two parameters: *Seroconversion time* is the time between infection and transition to immunity, and the *infection radius* indicates the distance at which other people are infected. Other parameters such as the probability of infection should be added. The simulation consists of 200 "persons" of only one species, symbolized by colored circles: Green for healthy, Red for infected and Yellow for immune. The individuals scurry around, encountering and infecting each other. After the seroconversion period, they become immune. *Hilberto* has little to do: he sets a few initial values and logs the percentage of infected at one-second intervals. If needed, he can generate a graph of the recorded data.



Somewhat more active is a *person*, so a sprite that serves as a template for the other 199 clones. Since these are supposed to be healthy at the beginning, it assumes the "healthy" costume, creates the clones, resets the timing and gives a start signal. It also changes to the "sick" costume. Thus it is the starting point of the infection.

The generated clones take a random position, set their speed to a random value and the previous disease duration to -1, because they are still healthy.

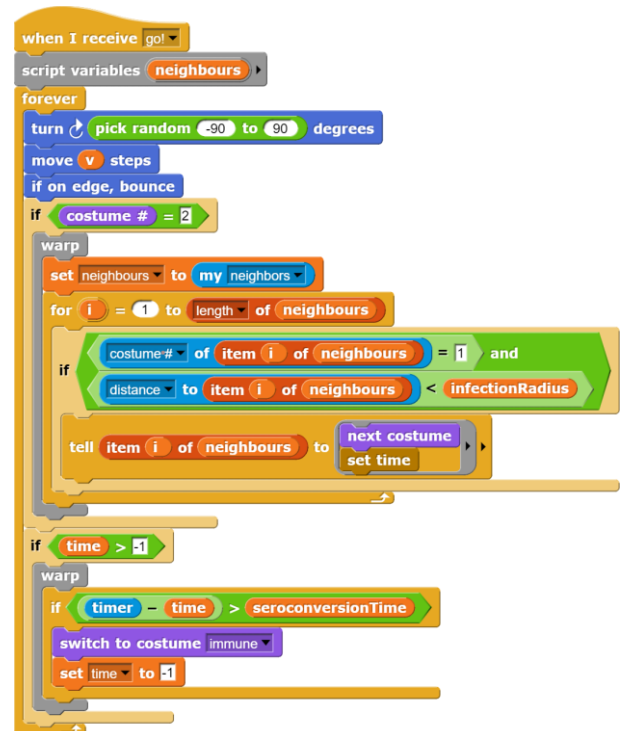
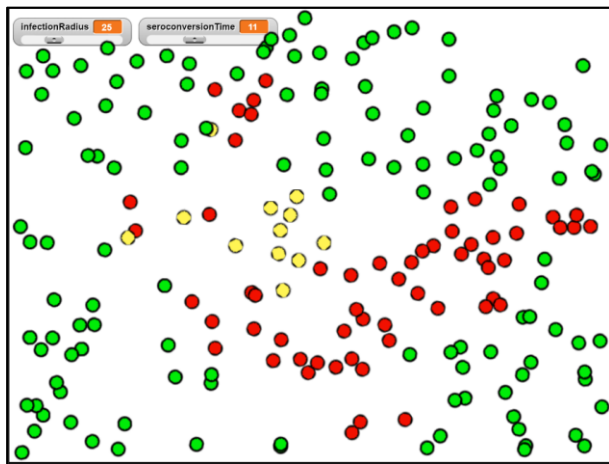


After receiving the start signal, a person turns a little and moves. If it hits the edge, then it bounces off it.

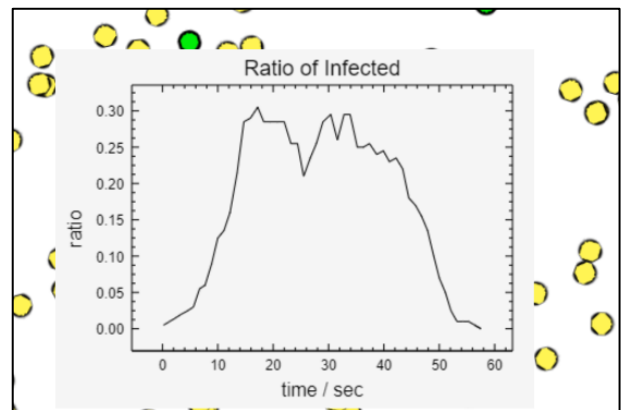
If the person is sick, then he determines his neighbors. If they are healthy and close enough, then they get sick.

If the person has been sick for a while, then see if the seroconversion time is up. If it is, she become immune.

That's it!



A typical diagram of the course of infection:



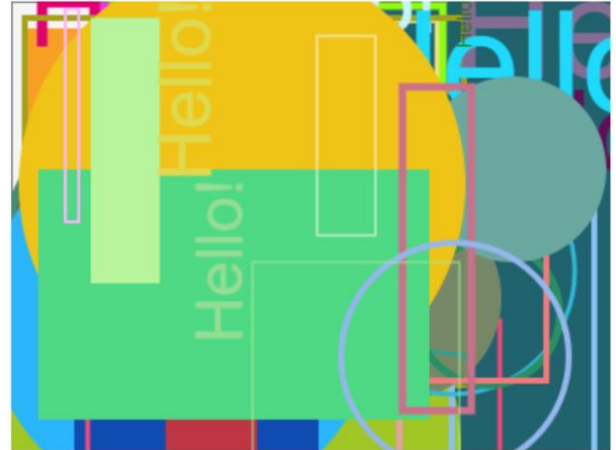
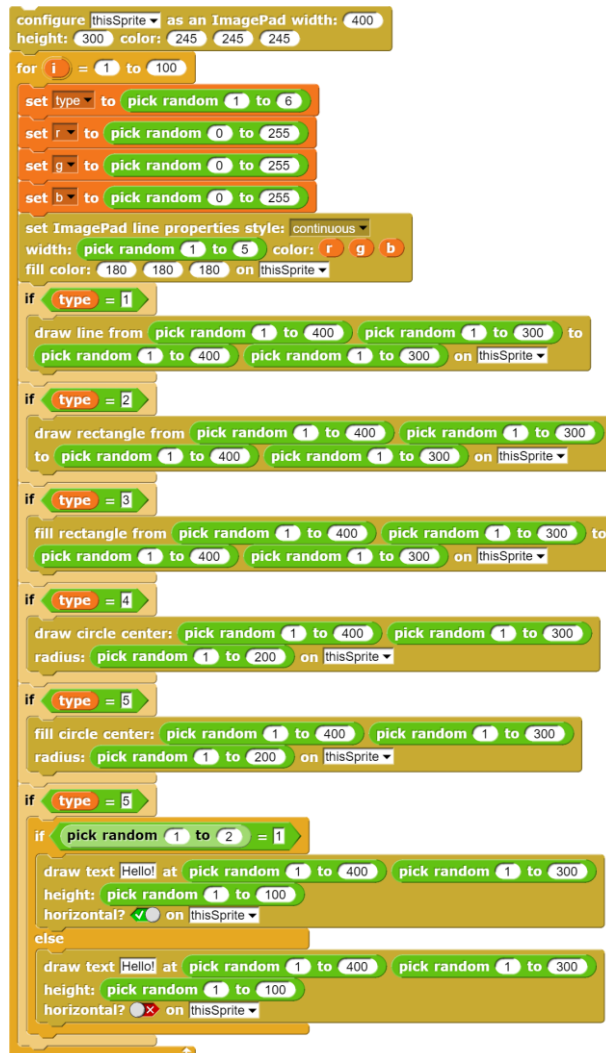
Tasks:

1. Persons do not fall ill every time they come into contact with diseased persons. Introduce a probability of infection.
2. There are different persons, e.g. those who wear protective masks - or not. Take this into account in the simulation.
3. People's mobility is also different, some stay mostly at home while others travel around the world. Take that into account in the simulation.
4. Viruses also evolve with time. Every now and then, mutations arise that cause a different likelihood of infection. Take this into account in the simulation.

8 Graphic related Examples

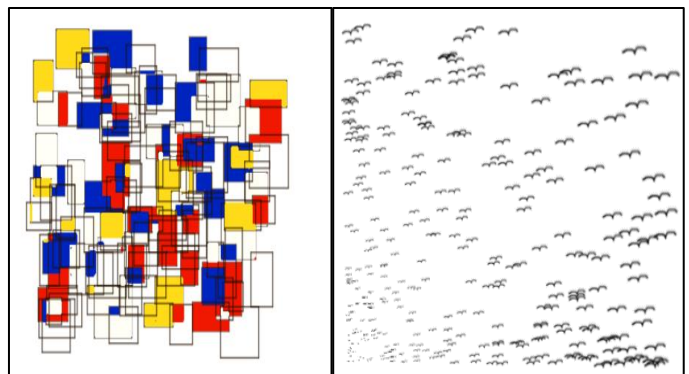
8.1 Simple random graphic

We simply draw 100 randomly chosen graphic elements on top of each other.



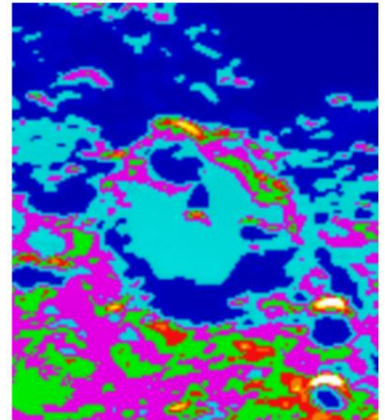
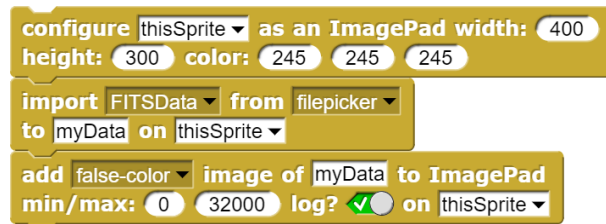
Tasks:

1. Search the web for images by *Piet Mondrian*. Try to create similar random images on the *ImagePad*.
2. Using a "vanishing point", you can create images in which objects appear to move "from back to front". Try it.



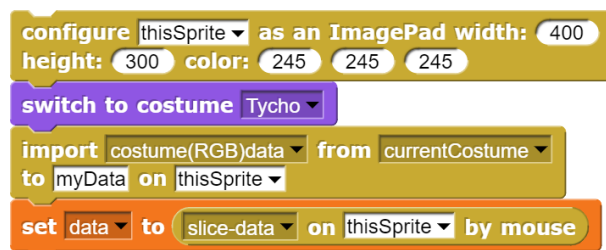
8.2 False color image of a lunar crater

We import the image data from a FITS file and then display it as a false-color image.

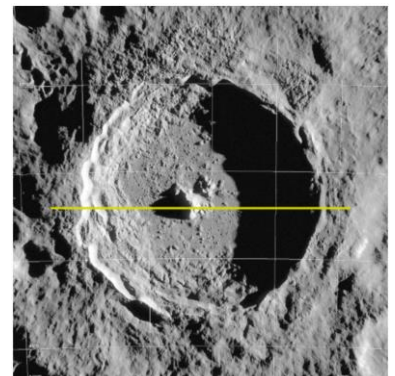


8.3 Slice through an image of the lunar crater Tycho

<https://www.spektrum.de/fm/912/thumbnails/Mond0.jpg.2996657.jpg>

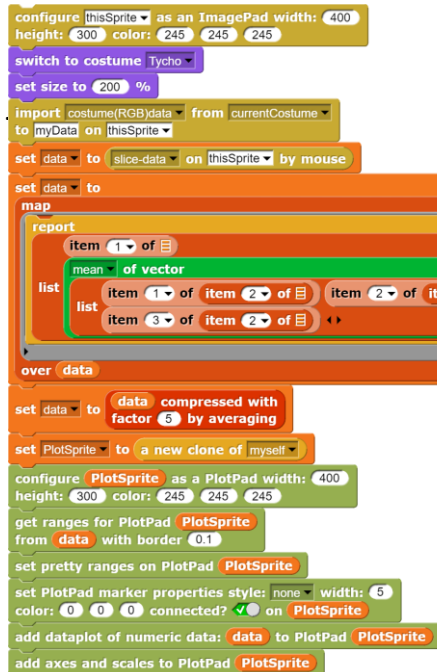


data		
376	A	B
1	0	
2	1	
3	2	
4	3	
5	4	
6	5	
7	6	



8.4 Shadow lengths in the lunar crater Tycho

We create an image of the lunar crater on an *ImagePad*, import its data and create a slice through the image using the mouse. We plot the data values of the section line on a *PlotPad*. From this and some additional data, the shadow lengths can be calculated.



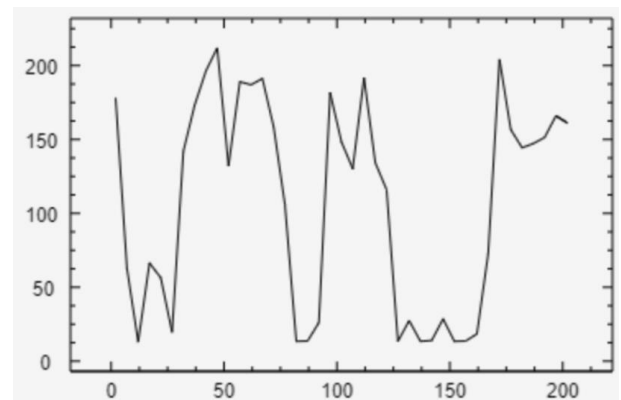
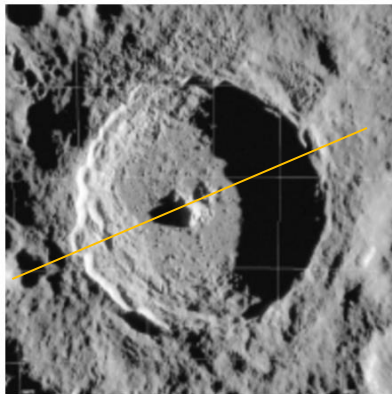
Representation of the crater image on the ImagePad.

Data recording with the mouse.

Conversion of RGB values to gray values.

Data compression with a factor of 5.

Generate a plot on a second sprite configured as a PlotPad.



8.5 Display of image data as histogram

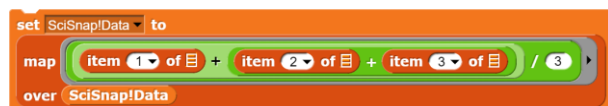
An RGB image is loaded, decomposed into grayscale, and the normalized distribution of the image values is displayed as a histogram on a new *PlotPad*. We find the actual image as the costume of an additional sprite called "ThePicture".

First of all we load the image into the data area *SciSnap!Data*:

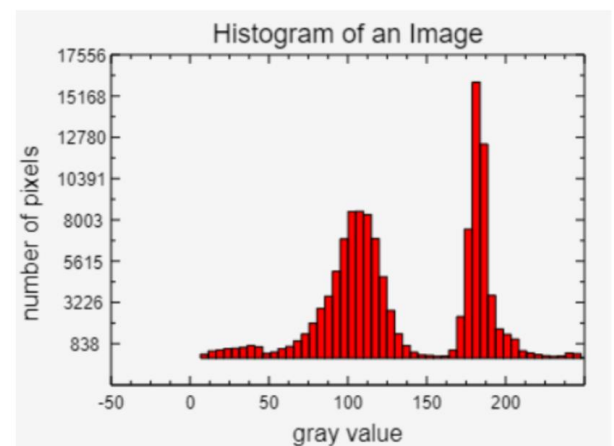
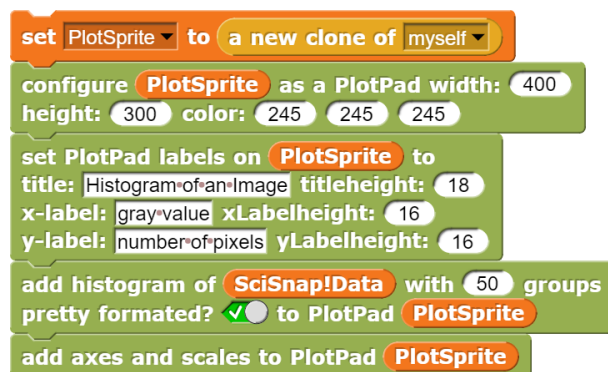


We obtain 120,000 RGB values.

We convert these into gray values.



Then we create a *PlotSprite* as a clone of the current sprite and configure it as a *PlotPad*. On this we create the histogram.



Tasks:

1. Search the web for different sets of data. Display them or parts of them graphically.
2. Automate histogram generation by adding a new block *histogram of <costume>*. Compare the histograms of typical image types. To what extent is it possible to compare images in this way, or where might difficulties arise?
3. In the same diagram, represent the three colors of an RGB image by graphs and/or histograms.

8.6 Simulation of a planetary transit in front of a sun

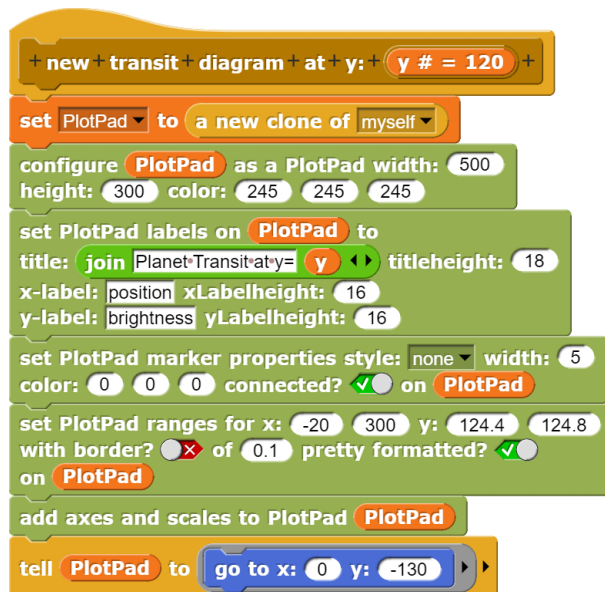
We look for a nice picture of the sun (source: [SchulAstro]) and load it as a costume of a sprite. To make it look more like outer space, we enlarge the stage and color it black. If we also draw the planet, we get the following picture.



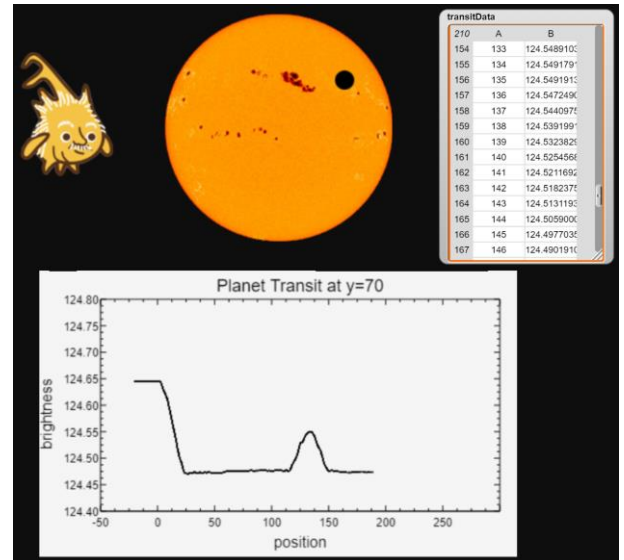
The planet should pass in front of the sun as a black circle. When we draw such a circle, we change the actual image of the sun. Therefore, we draw a copy *newCostume* from this image, on which we draw after that. Our planet should move from the very left a little bit outside of the image ($x=-2r$) to the very right ($x=\text{image width}+2r$) on the height y . We can also specify the radius r of the planet.

We can determine the current brightness of this arrangement without too many copying processes by subtracting the brightness of the pixels covered by the planet from the total brightness determined at the beginning. For this purpose, we import the image of the sun into the data area *myData* and determine the brightness around the center of the image in the radius "half image width" as well as the number of pixels involved. *brightness around* provides the summed gray value as well as this number. From these values we calculate the average brightness of the "slightly darkened" sun and store it together with the current position in the variable *transitData*.

Parallel to the transit, a diagram is to be created in which we can follow the results "live". For this purpose, we create another sprite, which we call *PlotPad* and which we configure accordingly. We wrap the necessary commands for this in a block called *new transit diagram*.



Then we write a block that determines the brightness data as described and refreshes the diagram in parallel. The result corresponds roughly to one of the methods used to find exo-planets.

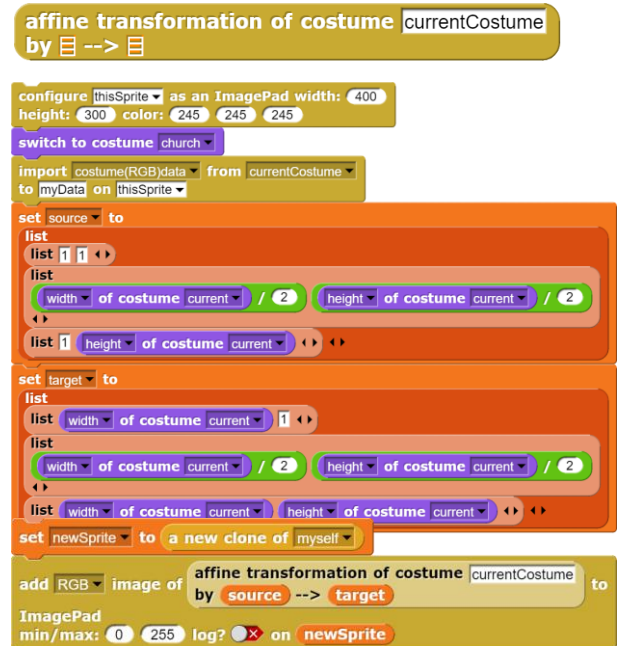


8.7 Affine transformation of an image

In the *ImagePad* library we find a block that allows to make affine transformations in an image by mapping three points to three others - and all other points accordingly. The current costume of a sprite is taken as the image.

We want to mirror an image vertically at the center line. We load the image - here: of a church - and select corresponding points at the edges. These combine them to the two lists *source* and *target*.

Finally, we create a clone of the *ImagePad* and ask it to display the transformed image as a costume.



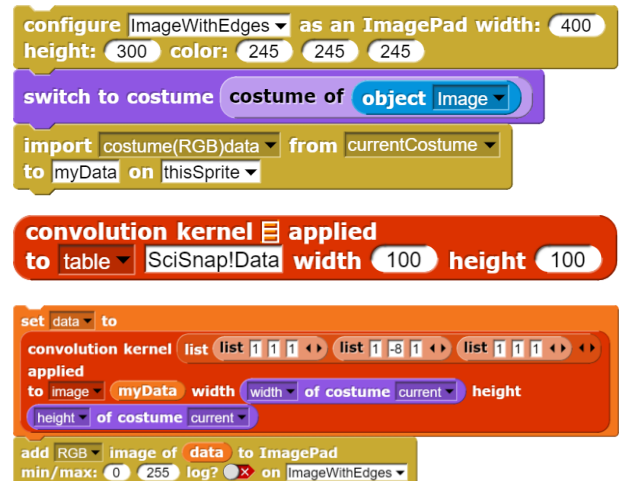
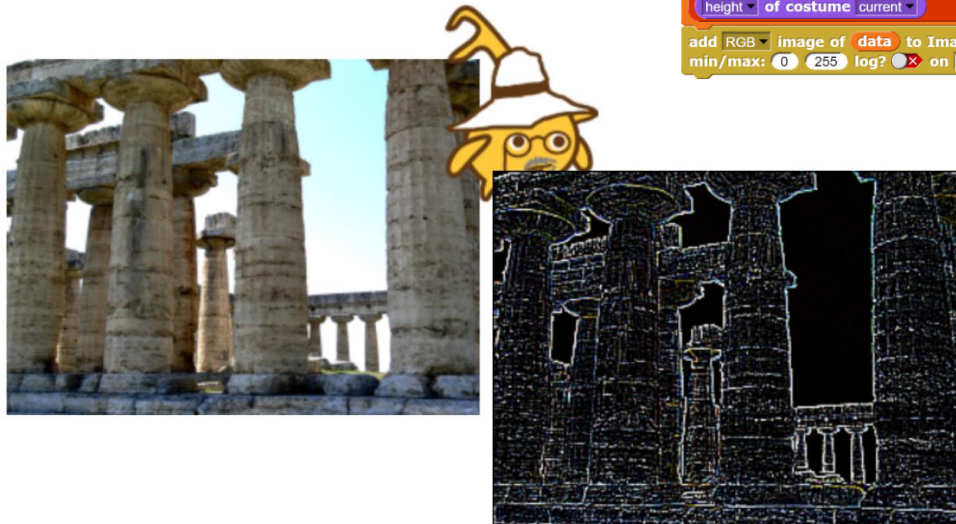
8.8 Kernel application for edge detection

We configure a sprite as an *ImagePad* and change to the costume of an ancient temple. We import this image into the *myData* data area of the *ImagePad*.

On this image data we apply the *Laplace* kernel

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

using the block *convolution kernel applied* of the *Data tools* and store the result in the variable *data*. We display the result as a new costume. For comparison, a second sprite represents the original image.



Tasks:

1. Images are sometimes a bit "flat". This is because they do not use the full range of values for the three color channels from 0 to 255.
 - a: Develop a method to determine and display the value range of an image.
 - b: Develop a method to exploit the full range of values, i.e. map black pixels to 0, bright pixels to 255.
 - c: Summarize the method in a new block, which is passed the costume of a sprite and returns the improved costume as result.
2.
 - a: On pictures, you can try to find "faces" by highlighting related areas of a color range, e.g. "orange", and erasing the rest of the picture. Try to develop a new block for this.
 - b: Using a kernel, the edges of such areas can be isolated. Find out about suitable kernels on the web, for example, and try them out for the purpose mentioned.
 - c: Faces are often "oval". Try to distinguish faces from other "orange" objects in this way.
3.
 - a: Really artistic photos are black and white, of course. If you don't have any, you can create grayscale images from RGB images. Do this.
 - b: It looks even more artistic if the photos are "hard", that is, have a very strong contrast. Experiment a bit!

8.9 Diffusion in the grid model

If you put some paint into a container of water, the paint particles slowly disperse into the surrounding water. This is called *diffusion*. We can imagine the process as an exchange process in which paint particles exchange places with water particles - and of course particles of the same kind also exchange places, but you can't see that.

In the grid model we symbolize water particles with the value for blue (9) and color with the value for red (4). We configure a sprite as *ImagePad* and initialize the grid with dimensions 400x400. After that all grid cells get the value 9 and then the ones in the center get the value 4. The grid can now be displayed - with a red "block of paint" in the center.

```
configure thisSprite as an ImagePad width: 400
height: 400 color: 245 245 245
set ImagePad grid properties on thisSprite
horizontal cells: 400 vertical cells: 400
fill all cells on thisSprite range x: 1 xMax y: 1 yMax
randomly with numbers 9
fill all cells on thisSprite range x: 175 225 y: 175 225
randomly with numbers 4
add grid myData on thisSprite with grid lines? ☒
```

After that, diffusion starts by randomly swapping the values of the grid cells with neighbors. For one cell this is a simple process, but for our 160,000 cells it is a bit complex. *SciSnap!* therefore has its own block, which executes the process of this "heat movement" for all cells n times:

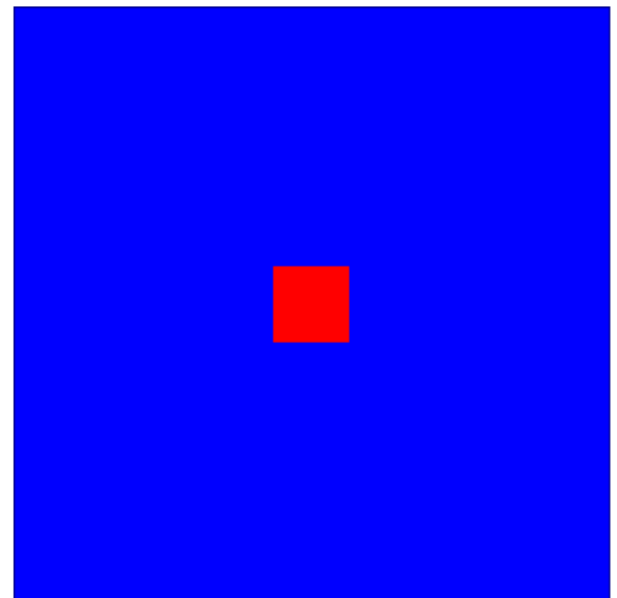
```
all cells on thisSprite as torus? ☒ swapped 1 times
randomly inside radius 1 range x: 1 xMax y: 1 yMax
```

We simply assign the result to the data area of the grid *myData* and redisplay the result. We repeat these operations again and again.

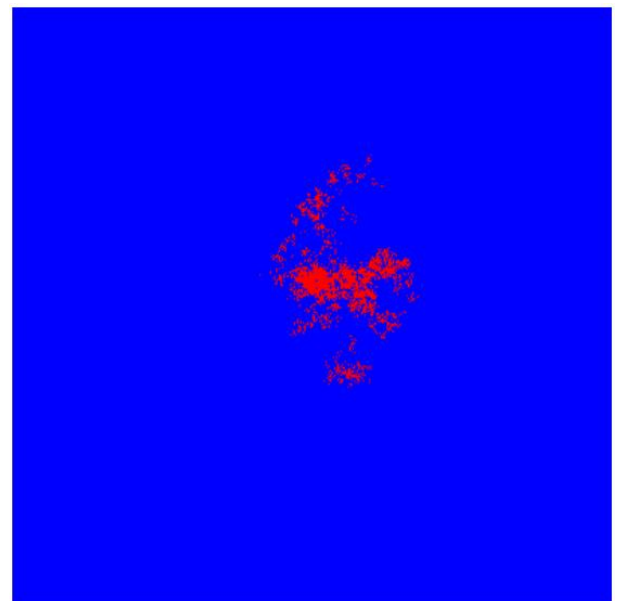
```
forever
  set myData to all cells on thisSprite as torus? ☒ swapped 50 times
  randomly inside radius 1 range x: 1 xMax y: 1 yMax
  add grid myData on thisSprite with grid lines? ☒
```

Overall our script is:

```
configure thisSprite as an ImagePad width: 400
height: 400 color: 245 245 245
set ImagePad grid properties on thisSprite
horizontal cells: 400 vertical cells: 400
fill all cells on thisSprite range x: 1 xMax y: 1 yMax
randomly with numbers 9
fill all cells on thisSprite range x: 175 225 y: 175 225
randomly with numbers 4
add grid myData on thisSprite with grid lines? ☒
forever
  set myData to all cells on thisSprite as torus? ☒ swapped 50 times
  randomly inside radius 1 range x: 1 xMax y: 1 yMax
  add grid myData on thisSprite with grid lines? ☒
```



And after a few cycles our color spot looks like this, for example:



8.10 Ferromagnetism as a Grid Automaton

Self-organization processes often occur in cellular automata due to the interaction of neighboring cells. We want to build a simplified version of the *Ising model*²⁴ of ferromagnetism. To do this, we imagine a grid of small elementary magnets, e.g., atoms with a magnetic moment, randomly oriented up or down. We encode the direction with the colors green (7) and red (4). So, the beginning of the script is very similar to the previous one.

Now we assume that elementary magnets are opportunists: their behavior simply follows the majority of their neighbors. If the majority of elementary magnets in the neighborhood is oriented upwards, then the small magnet will also be oriented upwards, otherwise it will be oriented downwards. Because such queries are common in lattice automata, there is an own *SciSnap!* block.

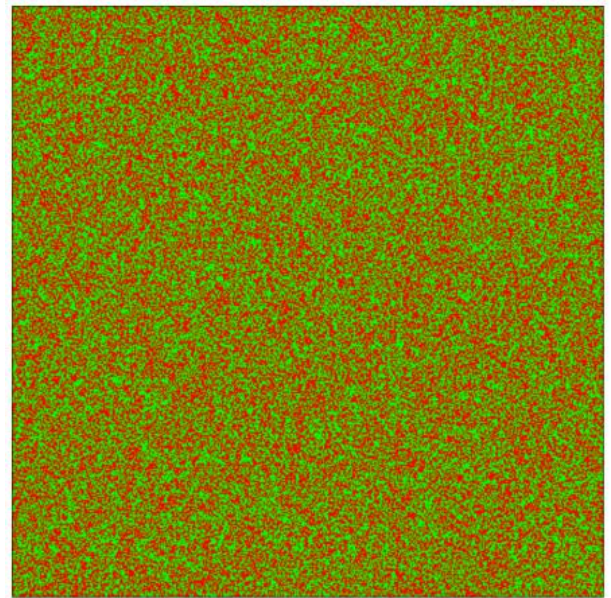
all cells on thisSprite as torus? ☒ with value any-or-number
take value 2 if number of surrounding value 2 is greater-than 4
else take value 1 with noise? ☒ of 5 %
range x: 1 xMax y: 1 yMax

The term "noise" is used to describe a noise that randomly "tilts" the specified percentage of cells.

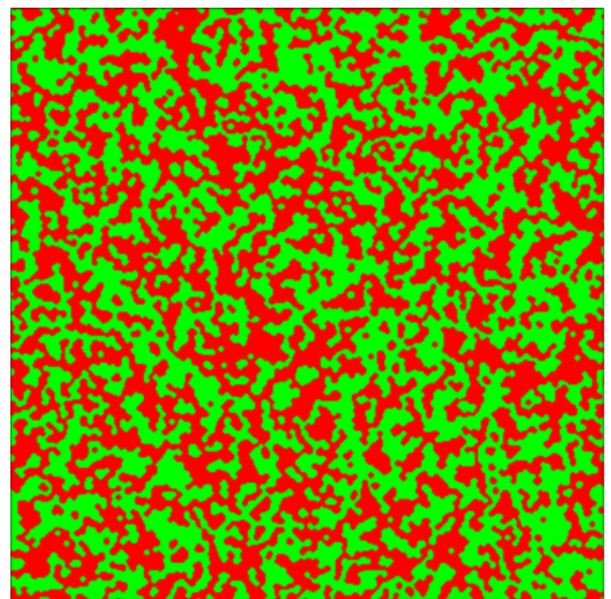
Thus, as in the last example, the new states of the lattice automaton can be repeatedly calculated and displayed. Altogether we get the following script and after some passes the displayed result, in which "intertwined" larger areas arise, which align themselves in an external magnetic field and strengthen it. The grid behaves like a ferromagnet. Strong noise, which physically corresponds to an increased temperature, disturbs the magnetization until it collapses at high values.

```
configure thisSprite as an ImagePad width: 400
height: 400 color: 245 245 245
set ImagePad grid properties on thisSprite
horizontal cells: 400 vertical cells: 400
fill all cells on thisSprite range x: 1 xMax y: 1 yMax
randomly with numbers 4 7
add grid myData on thisSprite with grid lines? ☒
forever
  set myData to
    all cells on thisSprite as torus? ☒ with value any
    take value 4 if number of surrounding value 4 is greater-than 4
    else take value 7 with noise? ☒ of 60 %
    range x: 1 xMax y: 1 yMax
  add grid myData on thisSprite with grid lines? ☒
```

```
configure thisSprite as an ImagePad width: 400
height: 400 color: 245 245 245
set ImagePad grid properties on thisSprite
horizontal cells: 400 vertical cells: 400
fill all cells on thisSprite range x: 1 xMax y: 1 yMax
randomly with numbers 4 7
add grid myData on thisSprite with grid lines? ☒
```



```
forever
  set myData to
    all cells on thisSprite as torus? ☒ with value any
    take value 4 if number of surrounding value 4 is greater-than 4
    else take value 7 with noise? ☒ of 60 %
    range x: 1 xMax y: 1 yMax
  add grid myData on thisSprite with grid lines? ☒
```



²⁴ <https://de.wikipedia.org/wiki/Ising-Modell>

8.11 Conway's Game of Life

A famous example of cellular automata is the *Game of Life* by John Horton Conway from 1970(!). It consists of a two-dimensional cellular automaton containing living cells (black) and dead cells (white). The game proceeds according to three rules:

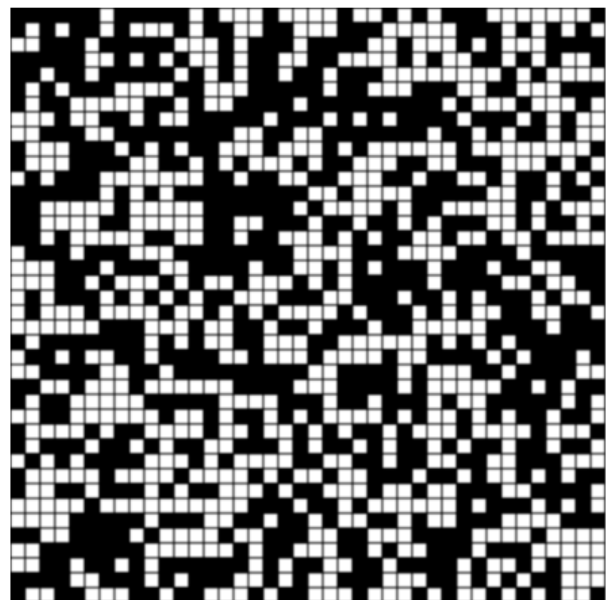
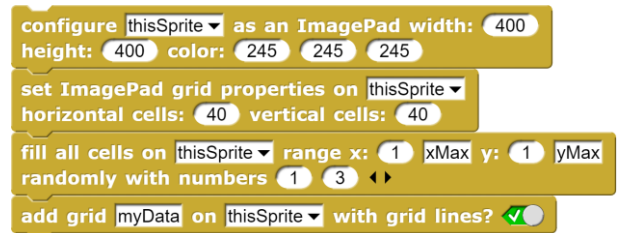
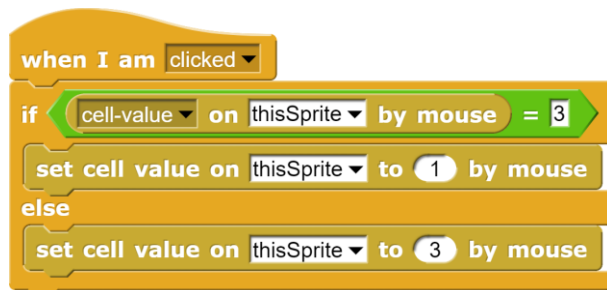
Rule 1: a white cell with exactly three black neighbors becomes black (alive).

Rule 2: black cells with less than two black neighbors die of loneliness, so they become white.

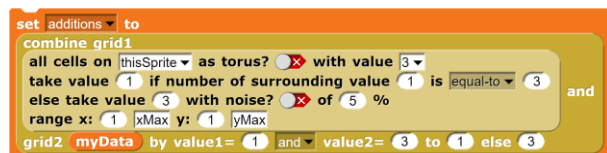
Rule3: black cells with more than three black neighbors die of overpopulation, so they become white.

Other cells do not change.

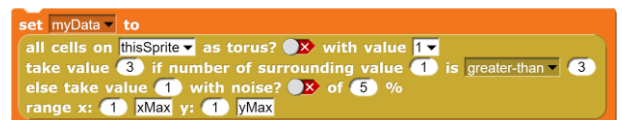
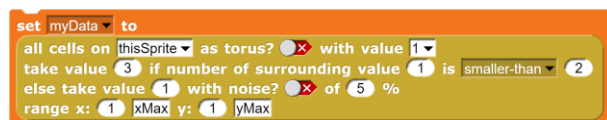
With the *SciSnap!* blocks for grids it is easy to create a Conway automaton. Instead of filling it randomly with values, we could also just create white cells and then change cells with the mouse:



Since the three rules should only have an effect on the following generation, we have to partially cache the results. We create a variable *additions* for this purpose. We fill this variable with the newly created cells. For this we apply rule 1 to the grid and subtract the old values:



The following are the "Robinson rule" 2 and the rule 3 for the "lemmings":



After the many dead, the new cells are added to the result:

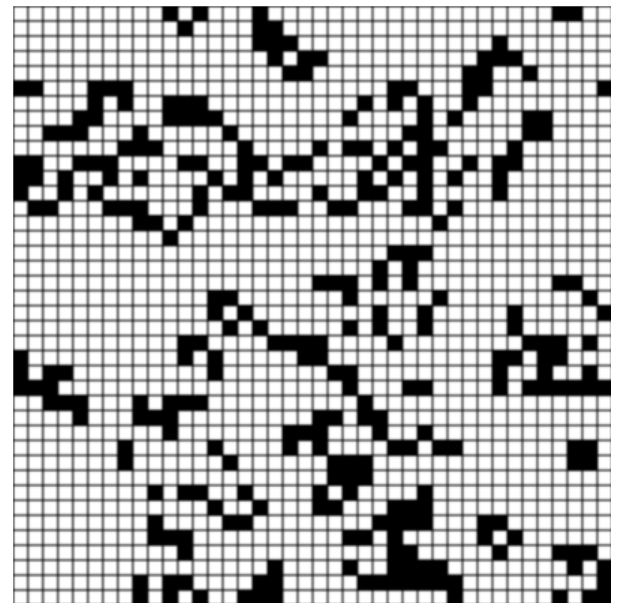
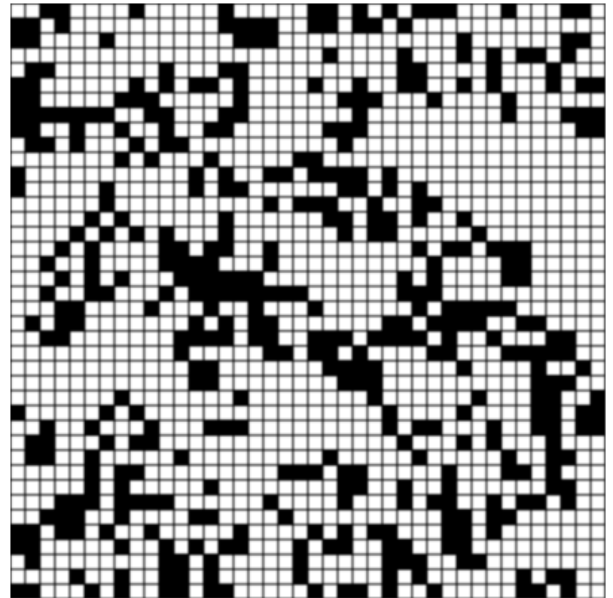


From this we build the overall script and run the game. After a few generations, stable patterns and objects that move are formed. Use your mouse to create patterns that you expect to be special (or search for them on the web). 😊

```

configure thisSprite as an ImagePad width: 400
height: 400 color: 245 245 245
set ImagePad grid properties on thisSprite
horizontal cells: 40 vertical cells: 40
fill all cells on thisSprite range x: 1 xMax y: 1 yMax
randomly with numbers 1 3
add grid myData on thisSprite with grid lines?
forever
  set additions to
  combine grid1
  all cells on thisSprite as torus? with value 3
  take value 1 if number of surrounding value 1 is equal-to 3 and
  else take value 3 with noise? of 5 %
  range x: 1 xMax y: 1 yMax
  grid2 myData by value1= 1 and value2= 3 to 1 else 3
  set myData to
  all cells on thisSprite as torus? with value 1
  take value 3 if number of surrounding value 1 is smaller-than 2
  else take value 1 with noise? of 5 %
  range x: 1 xMax y: 1 yMax
  set myData to
  all cells on thisSprite as torus? with value 1
  take value 3 if number of surrounding value 1 is greater-than 3
  else take value 1 with noise? of 5 %
  range x: 1 xMax y: 1 yMax
  set myData to
  combine grid1 myData and grid2 additions by value1= 1 or value2=
  1 to 1 else 3
  add grid myData on thisSprite with grid lines?

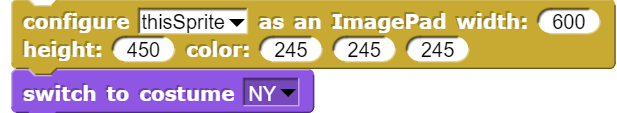
```



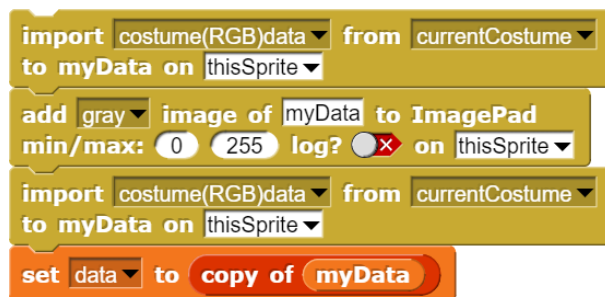
8.12 A cellular automaton as a soft focus device

In some cases, it is desirable to modify an image as a whole or in parts, e.g. to run a *soft focus* over it that creates an adjustable blur. One reason could be to make faces or car license plates unrecognizable. We will try this with our grid operations.

First of all, let's choose a nice picture, here of New York at night with dimensions 600x400, and represent it as a sprite costume.



Since our grid only takes simple numbers, we turn the image into a grayscale image, use it as a new costume, and import it back into *myData*. We save the result in the variable *data*.



Now we get really serious. We create a grid with the image dimensions, select the first column of the saved image data (the remaining two color columns are identical, after all) and transform it back into a matrix - with *reshape*. The result ends up in *myData*. Thus suitable grid values are available.



We can now apply one of the grid operations to this grid. We decide to take the mean values of the surrounding cell values in radius 4 as new cell values. This should create some blurring without making the image unrecognizable.



We now reassemble an image from the grid values by converting the grid back to a single column (with *reshape*), appending two identical columns to it and the transparency value per row 255. The result then corresponds to a *Snap!* pixel representation of a costume. We therefore display it directly on the sprite.

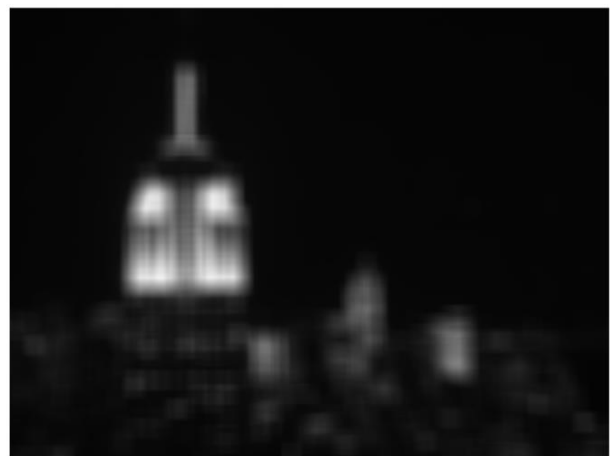
The entire script then is:

```

configure thisSprite as an ImagePad width: 600
height: 450 color: 245 245 245
switch to costume NY
import costume(RGB)data from currentCostume
to myData on thisSprite
add gray image of myData to ImagePad
min/max: 0 255 log? on thisSprite
import costume(RGB)data from currentCostume
to myData on thisSprite
set data to copy of myData
set ImagePad grid properties on thisSprite
horizontal cells: width of costume current vertical cells:
height of costume current
set myData to
reshape column first of data with first item? to
height of costume current width of costume current
set myData to
all cells on thisSprite as torus? take mean
of surrounding cells x: 1 xMax y: 1 yMax range: 4
set myData to
reshape myData to
width of costume current x height of costume current 1
set myData to
map
report list item 1 of item 1 of item 1 of 255
over myData
switch to costume myData
  
```



range 4, mean



range 10, mean



range 4, max

Tasks:

1. Scale the gray values to the range 0 to 9 (that of the nine grid colors) and display the image as a color image in the grid.
2. Find a picture with faces. Make only the faces unrecognizable.
3. Find a picture with car license plates. Make only the license plates unrecognizable.
4. Experiment a bit. Maybe it will become real art! 😊

8.13 Linear Wolfram automata

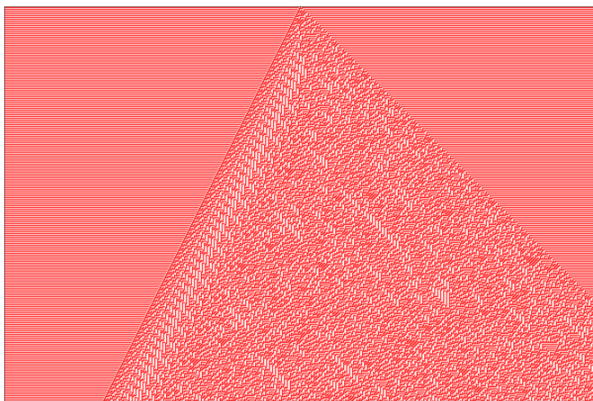
One of the most famous examples from the field of cellular automata are the linear automata of *Stephen Wolfram*²⁵. A linear grid automaton is a series of cells that can take two different values, which we can choose to be 0 and 1. Wolfram always considers three adjacent cells, constructs a dual number from the three cell values - which in this case can only be between 0 and 7 - and assigns a number to the subsequent state of the middle cell of this combination, which is 0 or 1, respectively. With these specifications there are $2^8=256$ different linear automata of this kind. The number of the automaton indicates directly its functionality. Wolfram draws below each other the respective assignment of the grid cells. The temporal order then runs from top to bottom and results in a two-dimensional grid. Inform yourself about the details of the processes elsewhere.

Since Wolfram automata are quite computationally expensive (and so interesting 😊), there is a separate block for them in *SciSnap!* With its help, we quickly create a script that generates and displays all Wolfram automata one after the other. As "seed" for the automaton we choose a single 1 in the middle of the first line. Enclosed are some results:

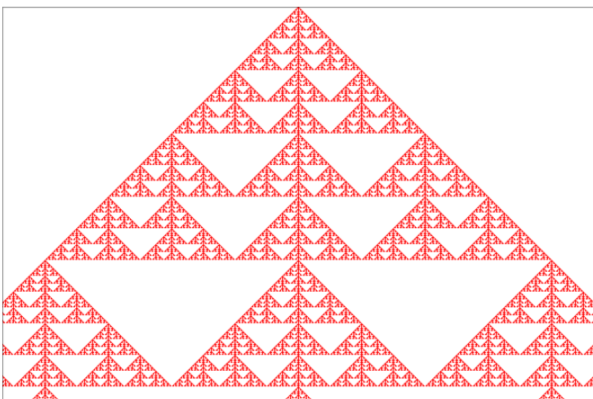
apply Wolfram automaton no 30 to grid with colors for 0: 3 and 1: 1

```
configure thisSprite as an ImagePad width: 600
height: 400 color: 245 245 245
set ImagePad grid properties on thisSprite
horizontal cells: 600 vertical cells: 400
for i = 1 to 255
  set n to join Wolfram-Automaton-No. i
  fill all cells on thisSprite range x: 1 xMax y: 1 yMax
  randomly with numbers 3
  set cell value on thisSprite at 300 1 to 4 with grid lines?
  add grid apply Wolfram automaton no i to grid myData on thisSprite
  with grid lines?
wait 0.5 secs
```

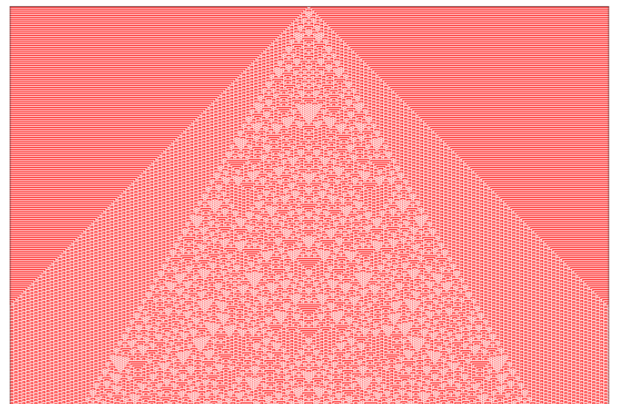
Wolfram Automaton No. 45



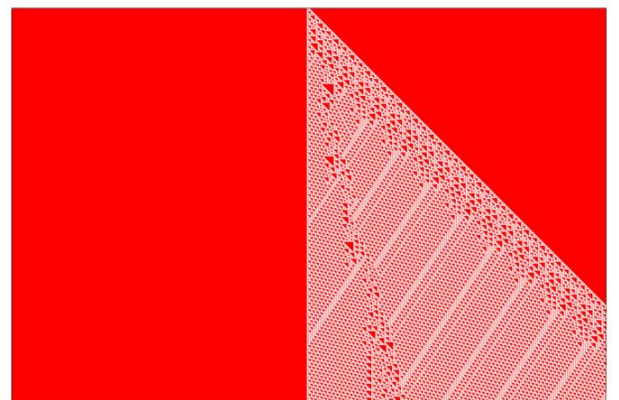
Wolfram Automaton No. 150



Wolfram Automaton No. 73



Wolfram Automaton No. 193



²⁵ [Wolfram]

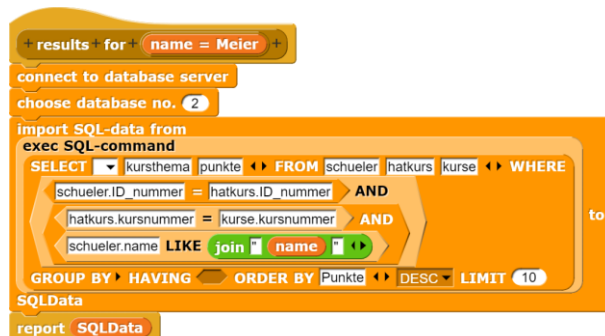
9 SQL Examples

9.1 Working with the SQL library

The *SQL tools* can be used to access either one of the sample databases or other data sources. The *configure SQL* block creates a global variable *SQLData* and sets some preferences. The block *connect to database server* connects to the server of the sample databases. If the sprite *Hilberto* with his costumes is available, then it changes into a server symbol and shows by a "lamp" whether the connection is established. If you want to use other servers, you have to change the server address and other settings in this block. With *read databases* you get a list of available databases. Select one of these with *choose database no. ...*

In practice you need the details of the tables of the used database all the time. If you assign the table attributes of a variable one after the other and display them with "*open in dialog*", you can place the corresponding table data in the *SciSnap!* window and start with the queries.

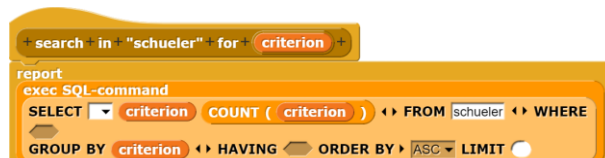
Example: The course title and rating for all of a learner's courses, sorted by grade in descending order, are searched for.



```

+ results for + name = Meier +
connect to database server
choose database no. 2
import SQL-data from
exec SQL-command
SELECT kurs thema punkte FROM schueler hatkurs kurse WHERE
schueler.ID_nummer = hatkurs.ID_nummer AND
hatkurs.kursnummer = kurse.kursnummer AND
schueler.name LIKE join name
GROUP BY HAVING ORDER BY Punkte DESC LIMIT 10
SQLData
report SQLData
  
```

Example: For statistical purposes, the *schueler* table is to be searched according to different criteria.




```

+ search in "schueler" for + criterion +
report
exec SQL-command
SELECT criterion COUNT ( criterion ) FROM schueler WHERE
GROUP BY criterion HAVING ORDER BY ASC LIMIT
  
```

configure SQL

connect to database server



```

1 snapex_example
2 snapex_school
3 snapex_world
4 snapextensions_db1
5 snapextensions_db2
6 test
length: 6
  
```

read databases

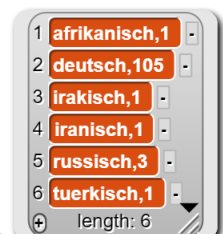
choose database no. 2

set data to attributes of table no. 3

6	items	3	items	4	items	5	items	19	items
1	snapex_example	1	hatkurs	1	ID_Nummer	1	KHJ	1	ID_Nummer
2	snapex_school	2	kurse	2	Kursnummer	2	kursthema	2	Name
3	snapex_world	3	schueler	3	Note	3	kurslehrer	3	Vorname
4	snapextensions_db1			4	Punkte	4	kursart	4	Geburtsort
5	snapextensions_db2					5	kursnummer	5	Geburtsort
6	test					6		6	Staatsang
								7	Geschlecht
								8	Konfession
								9	PLZ
								10	Ort
								11	Ortsteil
								12	an_der_BG
								13	Sekit seit

10	A	B
1	Das Thema :	14
2	Das Thema :	14
3	Das Thema :	14
4	Das Thema :	14
5	Das Thema :	13
6	Das Thema :	13
7	Abschluss de	13
8	Russisch	13
9	Philosophisc	11
10	Was soll ma	11

results for Arkson



```

1 afrikanisch,1
2 deutsch,105
3 irakisch,1
4 iranisch,1
5 russisch,3
6 tuerkisch,1
length: 6
  
```

search in "schueler" for Staatsang

9.2 Simple SQL query

We want to ask the topics of all English courses in a school database.

configure SQL

connect to database server

choose database no. 2

import SQL-data from
exec SQL-command

SELECT FROM WHERE LIKE

to SQLData

SQLData		
22	A	B
1	Problems an	En 31
2	Problems an	En 32
3	Landeskund	en 31
4	Fantasy and	en 32
5	Fantasy and	en 33
6	Landscapes	En 41
7	Landscapes	En 42
8	Shakespear	en 41
9	Education in	en 42
10	Education in	en 43
11	The Short St	En 11

9.3 More complex SQL query

Request to a school database: search for the best results in English.

configure SQL

connect to database server

choose database no. 2

import SQL-data from
exec SQL-command

SELECT AVG () FROM WHERE = AND LIKE

GROUP BY HAVING ORDER BY AVG () DESC

LIMIT

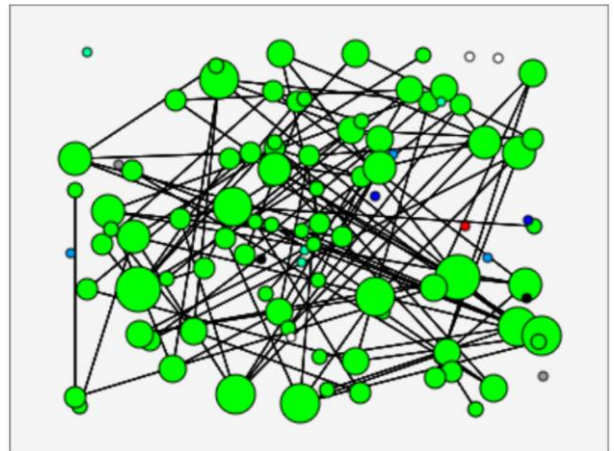
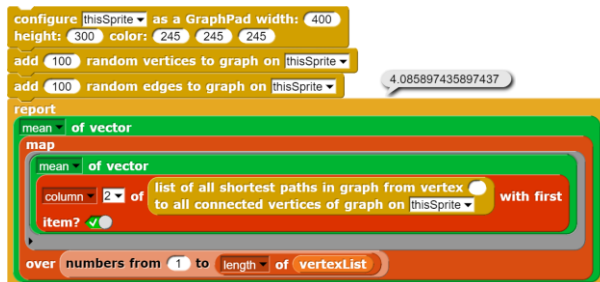
to SQLData

SQLData		
10	A	B
1	Rassin	12.5000
2	Schmitt	12.5000
3	Wuchtig	12.2500
4	Frugich	12.0000
5	Knusel	12.0000
6	Pfaffner	11.7500
7	Zinn	11.7500
8	Reinsberg	11.5000
9	Krahn	11.2500
10	Tohler	11.2500

10 Graph Examples

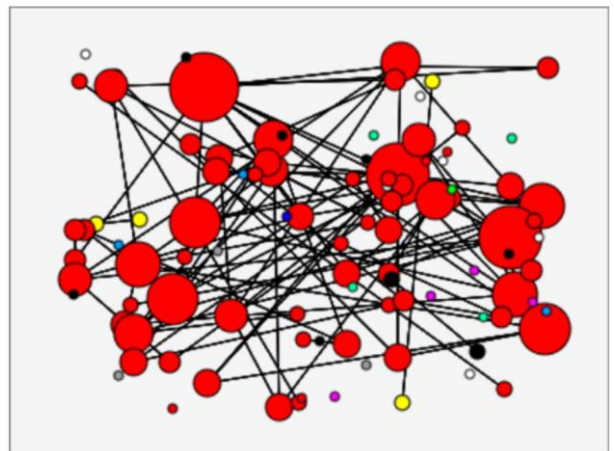
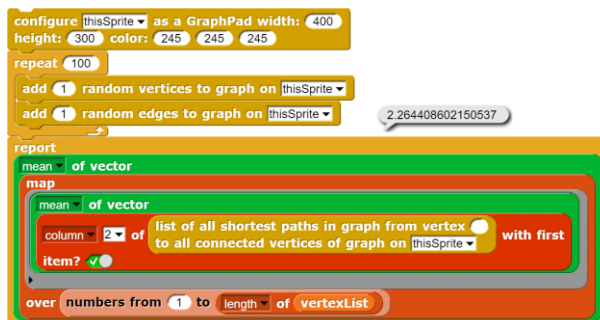
10.1 Mean distance in a random graph (small worlds)

We calculate the mean distance of the vertices in a graph where first all vertices and then all edges have been created.



10.2 Mean distance in a scalefree graph (small worlds)

We compute the average distance between the vertices in a graph where alternating vertices and edges have been created.

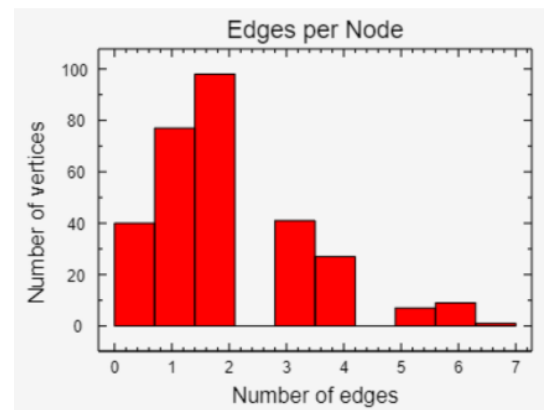
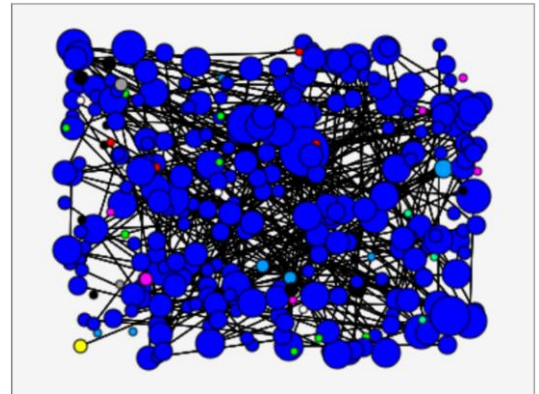


10.3 Edges per vertex histogram in a random graph

```

configure thisSprite as a GraphPad width: 400
height: 300 color: 245 245 245
add 300 random vertices to graph on thisSprite
add 300 random edges to graph on thisSprite
set PlotPad to a new clone of myself
configure PlotPad as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on PlotPad to
title: Edges per Node titleheight: 18
x-label: Number of edges xLabelheight: 16
y-label: Number of vertices yLabelheight: 16
add histogram of
map length of keep items [ ] from [ ] over with 10
adjacencyMatrix
groups
pretty formatted? to PlotPad PlotPad
add axes and scales to PlotPad PlotPad

```

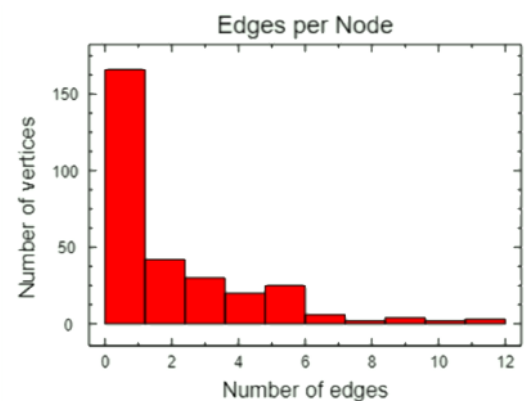
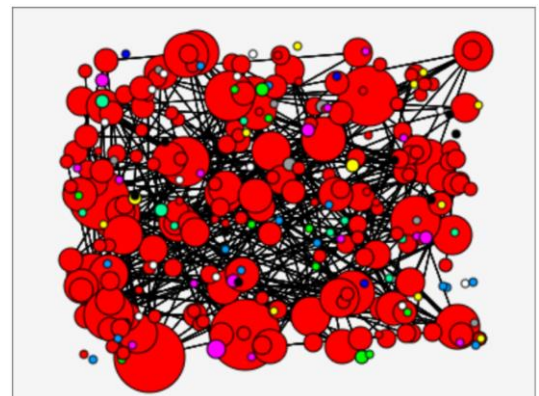


10.4 Edges per vertex histogram in a scalefree graph

```

configure thisSprite as a GraphPad width: 400
height: 300 color: 245 245 245
repeat 300
  add 1 random vertices to graph on thisSprite
  add 1 random edges to graph on thisSprite
set PlotPad to a new clone of myself
configure PlotPad as a PlotPad width: 400
height: 300 color: 245 300 245
set PlotPad labels on PlotPad to
title: Edges per Node titleheight: 18
x-label: Number of edges xLabelheight: 16
y-label: Number of vertices yLabelheight: 16
add histogram of
map length of keep items [ ] from [ ] over with 10
adjacencyMatrix
groups
pretty formatted? to PlotPad PlotPad
add axes and scales to PlotPad PlotPad

```



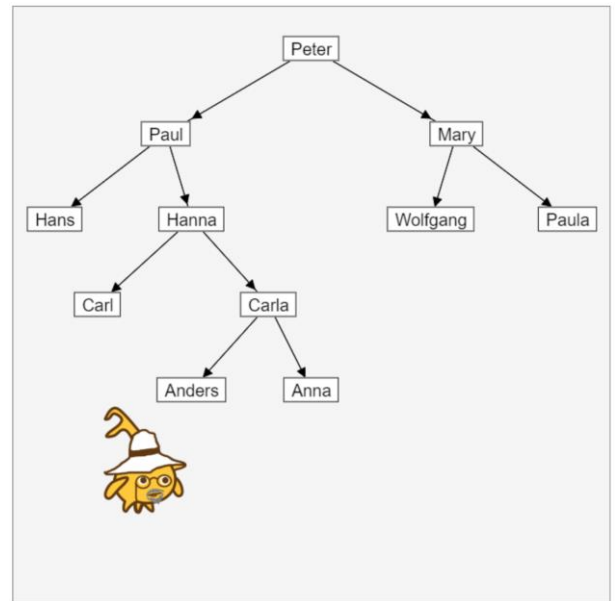
10.5 Breadth and depth search in a family tree

We create a family tree as a directed graph without edge weights simply by arranging and connecting the appropriate vertices. This is cumbersome, but simple. Just some fiddling.

```

+ new + family + tree +
configure FamilyTree as a GraphPad width: 700
height: 700 color: 245 245 245
set GraphPad vertex properties minSize: 10
growing? ☐ showsContent? ☒ on FamilyTree
set GraphPad edge properties lineWidth: 1
color: 0 0 0 directed? ☒ weighted? ☐
showsWeight? ☐ on FamilyTree
new vertex at 0 300 content: Peter on graph of FamilyTree
new vertex at -170 200 content: Paul on graph of FamilyTree
add edge from vertex 1 to vertex 2 to graph on FamilyTree
new vertex at 170 200 content: Mary on graph of FamilyTree
add edge from vertex 1 to vertex 3 to graph on FamilyTree
new vertex at -300 100 content: Hans on graph of FamilyTree
add edge from vertex 2 to vertex 4 to graph on FamilyTree
new vertex at -140 100 content: Hanna on graph of FamilyTree
add edge from vertex 2 to vertex 5 to graph on FamilyTree
new vertex at 140 100 content: Wolfgang on graph of FamilyTree
add edge from vertex 3 to vertex 6 to graph on FamilyTree
new vertex at 300 100 content: Paula on graph of FamilyTree
add edge from vertex 3 to vertex 7 to graph on FamilyTree
new vertex at -250 0 content: Carl on graph of FamilyTree
add edge from vertex 5 to vertex 8 to graph on FamilyTree
new vertex at -50 0 content: Carla on graph of FamilyTree
add edge from vertex 5 to vertex 9 to graph on FamilyTree
new vertex at -140 -100 content: Anders on graph of FamilyTree
add edge from vertex 9 to vertex 10 to graph on FamilyTree
new vertex at 0 -100 content: Anna on graph of FamilyTree
add edge from vertex 9 to vertex 11 to graph on FamilyTree

```



In this tree we can now solve all kinds of tasks. For example, we could ask whether a person is an ancestor of another. For this we need the number of the start node, which we determine with `vertexnumber of Peter in graph of FamilyTree`. The rest is done either by the block for the breadth search or the block for the depth search.

```

+ is + parent = Carla + ancestor of + child = Mary + ? +
script variables result
set result to
breadth first search of content parent
starting at vertex vertexnumber of child in graph of FamilyTree of graph
on FamilyTree
draw graph on FamilyTree
report result

```

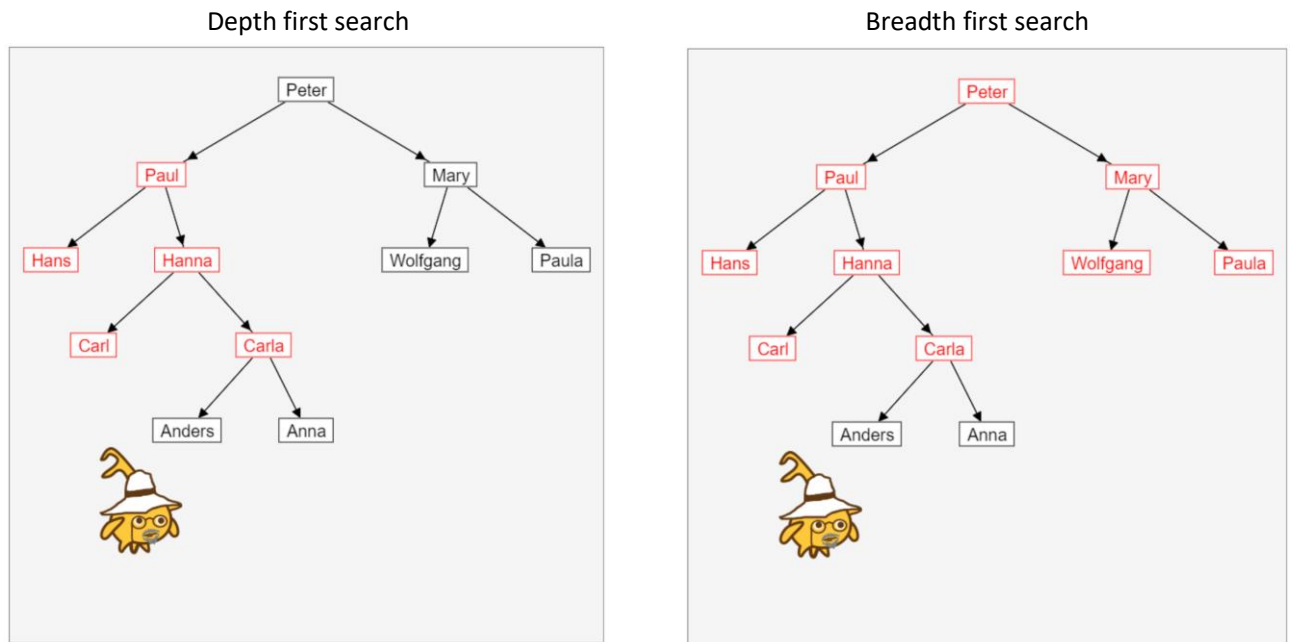
```

1 true
2 Carla found in vertex 9
length: 2

```

is Carla ancestor of Peter ?

In breadth first and depth first search, the visited vertices are marked and become red when the graph is re-drawn. This shows the differences of the two search methods quite clearly.



Tasks:

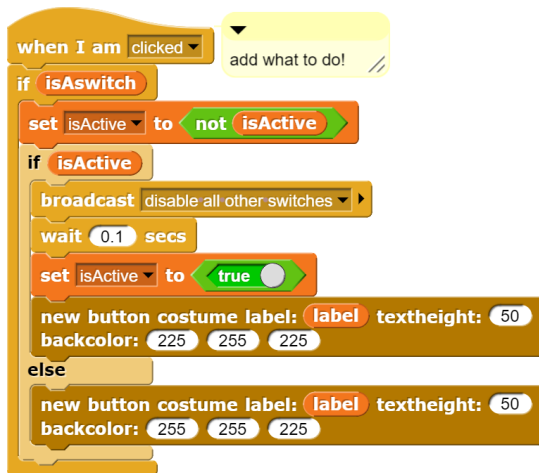
1. Determine the number of generations the relationship exists, if it exists at all.
2. Make lists of a person's relatives, e.g., parents, grandparents, aunts, brothers-in-law 😊, ...
3. a: Develop a script for creating a decision tree, e.g. for classifying animals or plants: In each case, it is asked whether it is a particular specimen or, if not, what question can be used to distinguish the named one from the current one ("Does it have four legs?"). Either the specimen or the question is entered into the tree.
 b: Let others test your result. Try to estimate from what amount of data in the tree it would be sensible to use such a program.
 c: Decision trees play a role in certain applications of machine learning (*Decision Tree Classification*). Learn about the method and its applications.

10.6 A graph lab

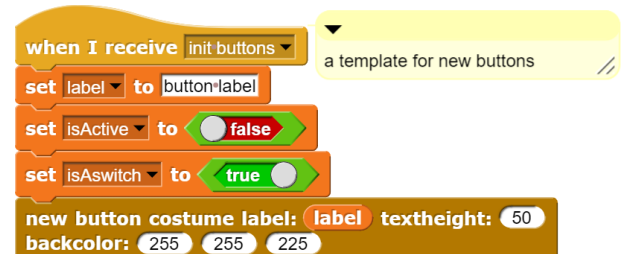
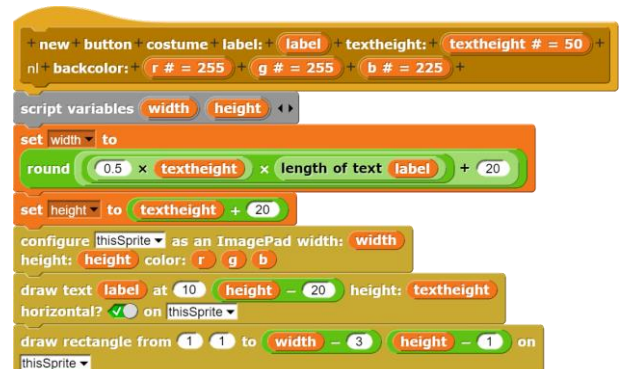
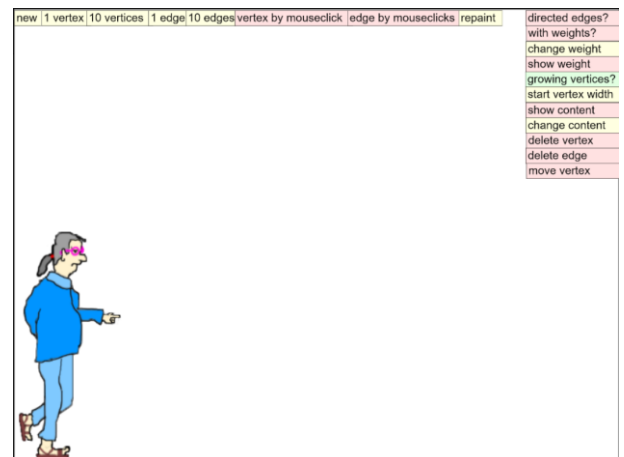
To be able to experiment with graphs easily, we build a small graph lab that allows to set the parameters of the graph - i.e. to be weighted, directed, ... - directly. The lab uses the stage as a *GraphPad*. Everything is controlled by *Gundolf de Jong*, the gifted young mathematician.

We want to insert a few buttons to be used either as a simple button or as a switch. To make it not too complicated, we build a simple template. Each button has an inscription, can be activated (or not) and can serve as a switch (or not). According to these specifications, a costume of the right size can be created, which the button, configured as *ImagePad*, will display. For the buttons, the possibility to move them (*draggable*) should be turned off.

The activity of the button is largely realized by calls to the *GraphPad* blocks. These are simply attached to the corresponding script. On mouse clicks on the stage, it then reacts to the presets.



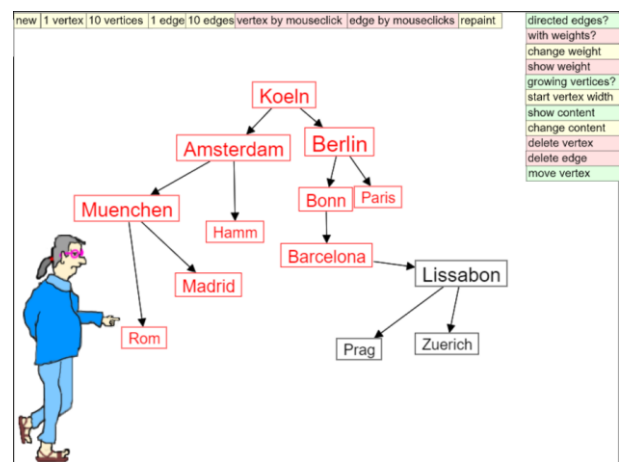
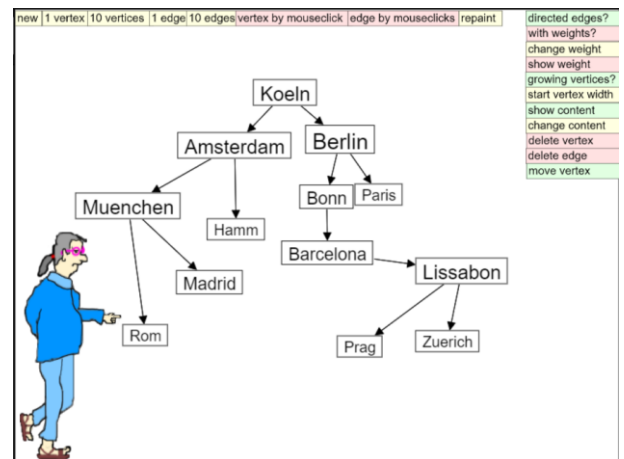
For example, if the button for inserting nodes is activated in the place of a mouse click, then the stage can cause the appropriate thing to happen.



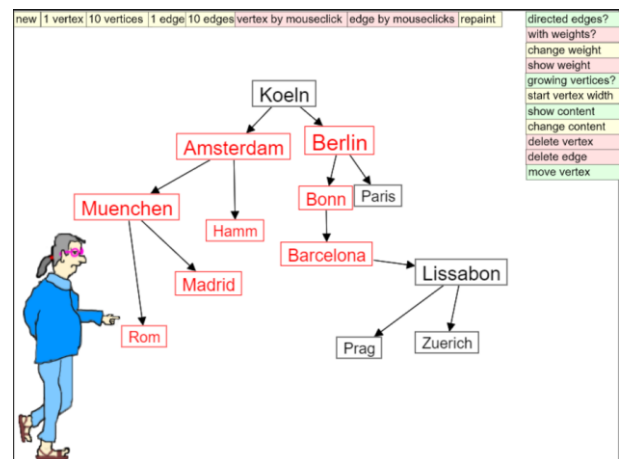
10.7 Tree search in the graph lab

We set the parameters for a tree: *directed edges*, *starting node size 10*. Then we position some nodes using the mouse (*vertex by mousedown*), enter their contents (*change content*), which are not ordered, and then arrange the nodes with the mouse so that the edges are visible and the contents are readable (*move vertex*).

In this tree, starting from the root, we search for the node with the content *Barcelona*, once by breadth first search, then by depth first search. To do this, we call the corresponding block and repaint the tree. The nodes marked during the traverse are then displayed in red. With the help of the block remove all markers of Graph they can be removed again.



The same with depth first search:

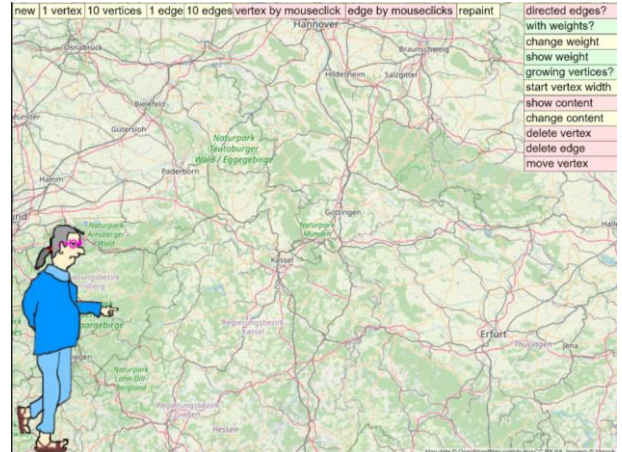
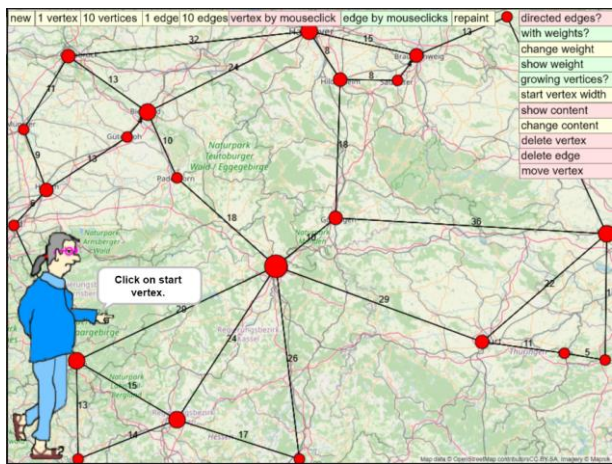


10.8 The graph lab with world map library

We load the Word Map Library, set the current location and a suitable zoom factor. On it, we want to create the graph of the closer surroundings, in which we will determine the most favorable paths between the locations. The graph should not be directed, but it should roughly show the relative distances. We get the following picture.



In this we first click on the places that are relevant for our traffic network (*vertex by mousedown*), especially the cities. Then we create the desired edges (*edge by mousedown*).



Where we can of course also display the vertex numbers (*show content*).

With the help of the graph, various questions can now be answered:

- How far is it (in our units) from Braunschweig (#4) to Siegen (#14)?

shortest path in graph from vertex 4 to vertex 14 on theStage

69

- How far is Hannover (#21) from Erfurt (#17)??

distance on theStage from vertex 21 to vertex 17

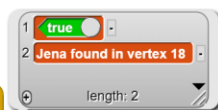
47

- After we have entered at least one city name ...

change content of vertex 18 to Jena of graph on theStage

... we can start a breadth first search from Münster (#7).

breadth first search of content Jena starting at vertex 1 of graph on theStage



- Where to go from Jena (#18)?

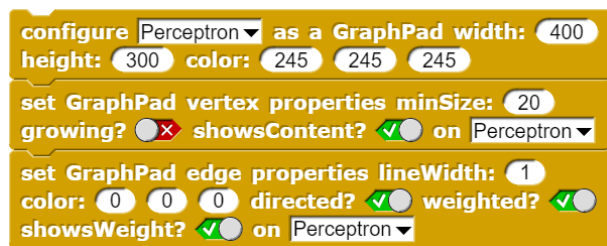
list of all shortest paths in graph from vertex 18 to all connected vertices of graph on theStage

22	A	B
1	1	40
2	2	50
3	3	68
4	4	67
5	5	71
6	6	81
7	7	92
8	8	85
9	9	89
10	10	72
11	11	68
12	12	58
13	13	64
14	14	69
15	15	78
16	16	66
17	17	11
18	18	0
19	19	5
20	20	22
21	21	76
22	22	54

11 Machine Learning Examples

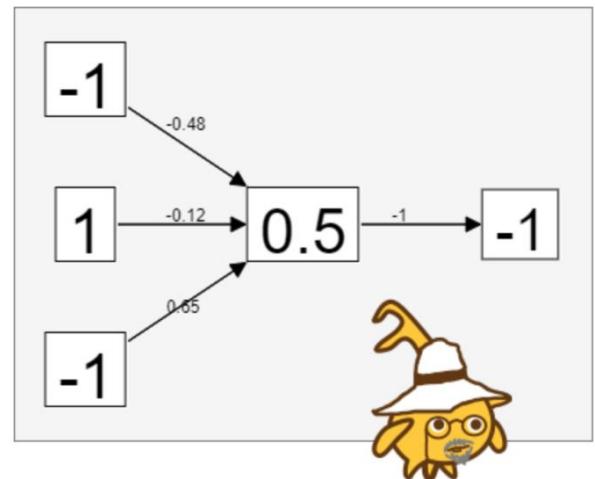
11.1 A simple perceptron as a graph

We want to ask *Hilberto* to use a *GraphPad* to illustrate how a simple *Perceptron*²⁶ works. To do this, he is to place three input nodes on the left of the image. in the middle sits the actual perceptron with a jump function that transmits either +1 or -1 to the output neuron on the right of the image. All in all, this is a directed graph with edge weights. *Hilberto* sets this up quickly: he creates a new sprite called *Perceptron* and configures it properly.



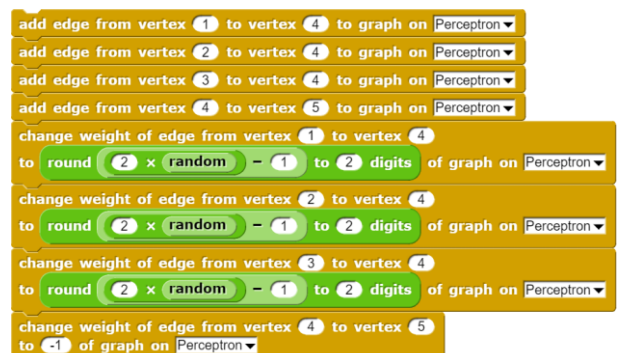
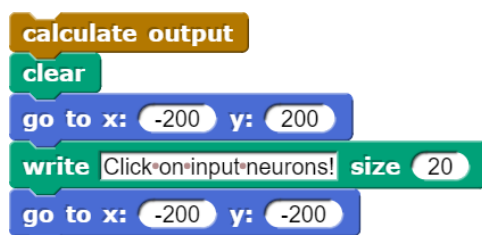
Then he adds specified vertices of the perceptron net.

Click on input neurons!



Only the edges are missing, first with random weights.

Hilberto assembles the blocks into a script and labels the whole thing, and of course he ensures a consistent situation by letting the perceptron compute through once.

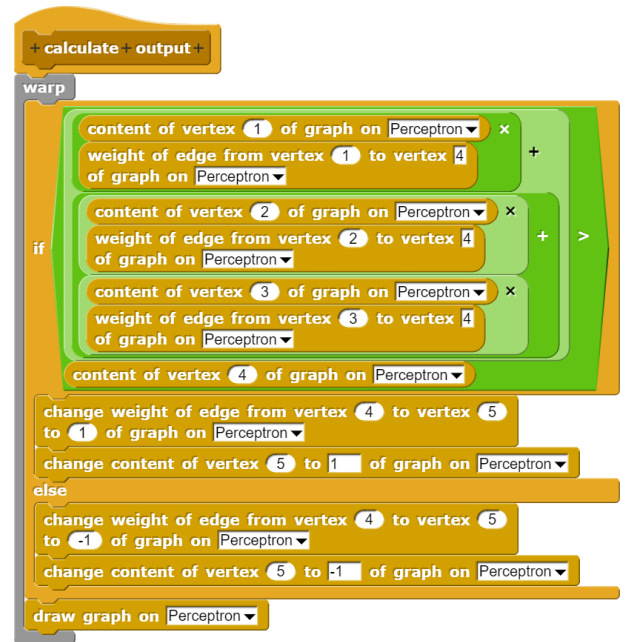
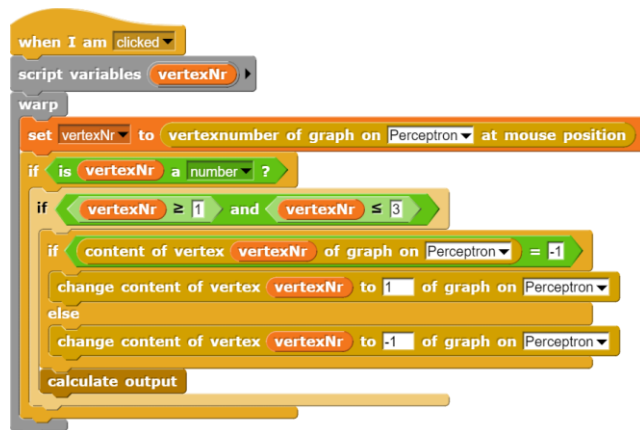


²⁶ <https://en.wikipedia.org/wiki/Perceptron>

The output of the network is determined by multiplying the values of the input neurons by the corresponding edge weights and summing the results. After that the result is compared with the value of the jump function in neuron 4. Depending on the result, the value of the last edge and the value of the output neuron is set.

However, the perceptron is also supposed to work in the following way: when an input neuron, i.e. a vertex on the left side, is clicked, it is supposed to change its value. To do this, we convert the mouse coordinates when clicked into sprite coordinates and then ask for the node number. If it is one of the three input neurons, then we change its value - and let recalculate.

Ready.



Hilberto himself is amazed that it can be done so easily.



Tasks:

1. Add a way to change the value of the jump function of the central neuron.
2. Add a way to change the edge weights of the three input neurons to the central neuron.
3. Change the edge weights and/or the jump function so that the perceptron works as an AND.
4. Change the edge weights and/or the jump function so that the perceptron works as an OR.
5. Change the edge weights and/or the jump function so that the perceptron works as an NAND.
6. Change the edge weights and/or the jump function so that the perceptron works as an NOR.
7. Can the perceptron also work as an XOR? Try it!
8. Search the literature for reasons why some circuits can be well realized by perceptrons and others not.

11.2 A simple learning perceptron

We now change the configuration a bit to teach the perceptron how to learn. To do this, we introduce another vertex, a "target vertex", under the output vertex, which displays "the correct" results, which in turn can be changed by clicking on it. If there is a difference between the values of the output vertex and the target vertex, the weights are changed until the "correct" result is obtained.

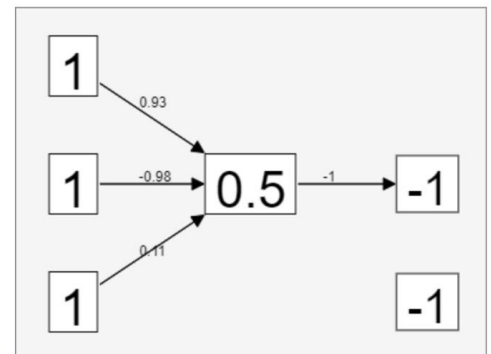
What to change?

When creating the net, only the new vertex must be inserted.

```

start SciSnap!
configure Perceptron as a GraphPad width: 400
height: 300 color: 245 245 245
set GraphPad vertex properties minSize: 20
growing? ☒ showsContent? ☒ on Perceptron
set GraphPad edge properties lineWidth: 1
color: 0 0 0 directed? ☒ weighted? ☒
showsWeight? ☒ on Perceptron
new vertex at -150 100 content: 1 on graph of Perceptron
new vertex at -150 0 content: 1 on graph of Perceptron
new vertex at -150 -100 content: 1 on graph of Perceptron
new vertex at 0 0 content: 0.5 on graph of Perceptron
new vertex at 150 0 content: -1 on graph of Perceptron
new vertex at 150 -100 content: -1 on graph of Perceptron
add edge from vertex 1 to vertex 4 to graph on Perceptron
add edge from vertex 2 to vertex 4 to graph on Perceptron
add edge from vertex 3 to vertex 4 to graph on Perceptron
add edge from vertex 4 to vertex 5 to graph on Perceptron
change weight of edge from vertex 1 to vertex 4
to round (2 * random - 1) to 2 digits of graph on Perceptron
change weight of edge from vertex 2 to vertex 4
to round (2 * random - 1) to 2 digits of graph on Perceptron
change weight of edge from vertex 3 to vertex 4
to round (2 * random - 1) to 2 digits of graph on Perceptron
change weight of edge from vertex 4 to vertex 5
to -1 of graph on Perceptron
calculate output
clear
go to x: -200 y: 200
write Click on input or target-value neurons! size 20
go to x: -200 y: 170
write Click on sprite to learn! size 20
go to x: -200 y: -200
  
```

Click on input or target-value neurons!
Click on sprite to learn!

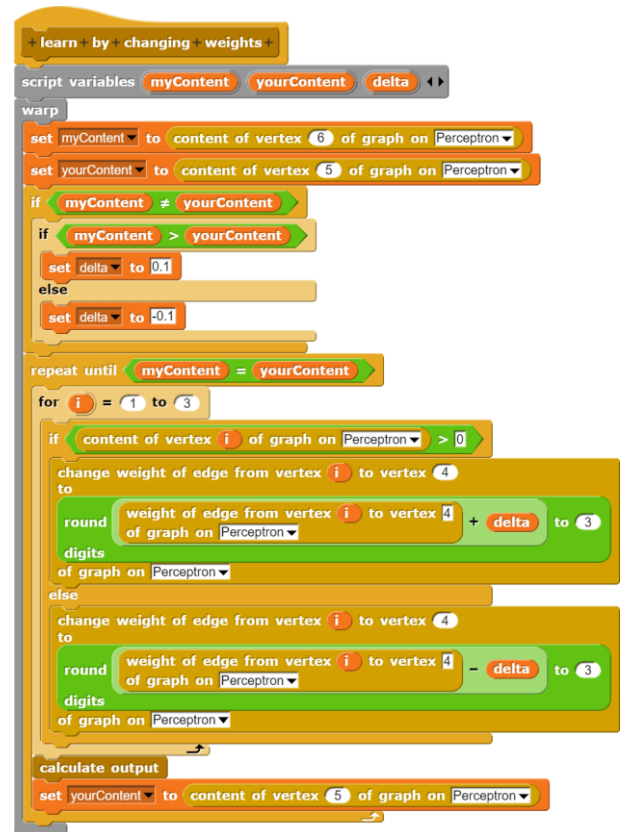


And at the click event on the *GraphPad* it has to be checked if a vertex or the pad was hit. In the second case, learning starts.

And how is learned?

If the values of the two neurons on the right of the image differ, then we change the weights on the edges of the input neurons until the correct result is obtained.

More precisely: We give a value to each of the three input neurons. Then, we set the desired value on the target neuron by clicking on it as well. Finally, we click anywhere on the *GraphPad* and watch it learning.



Tasks:

1. Add a way to implement learning by making changes to the value of the jump function in the inner neuron. Does this always work?
3. Train the network so that the perceptron works as an AND.
4. Train the network so that the perceptron works as an OR.
5. Train the network so that the perceptron works as an NAND.
6. Train the network so that the perceptron works as an NOR.
7. Can the perceptron also work as an XOR? Try it!

11.3 Training of a neural network

A neural network of width 4 with two layers (plus input layer) is generated and trained to deliver as output a 1 on the left and a -1 on the right, with zeros in between, when the vector of numbers 1 to 4 is input. It is to be noted that not the exact output values, but on the left the largest and on the right the smallest must be supplied.

The output of the net before training:

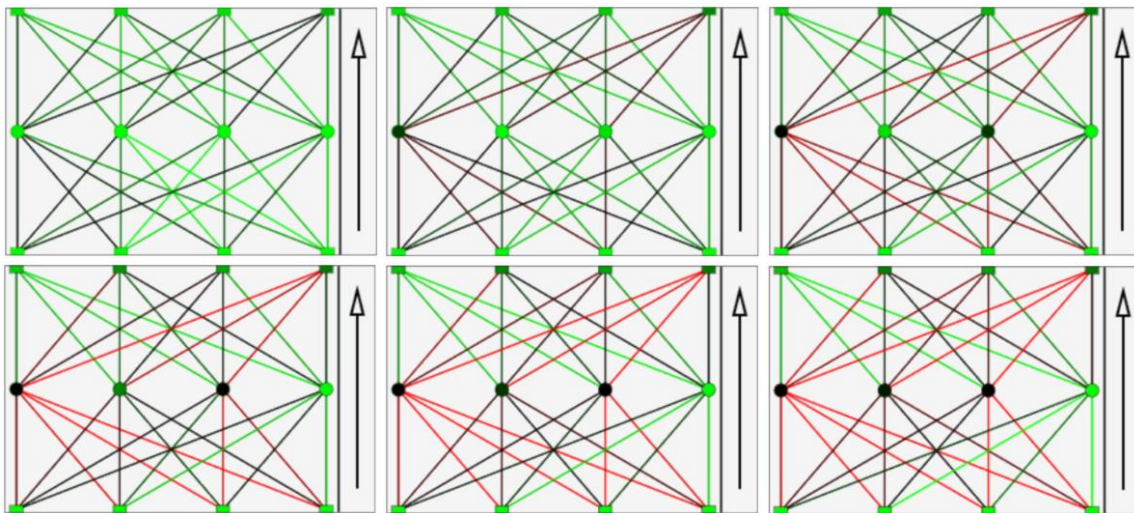
NN output of last layer with input numbers from 1 to 4 on thisSprite

```
start SciSnap!
configure thisSprite as a NeuralNetPad width: 300
height: 200 color: 245 245 245
NN add new weights for 2 layers of width 4 on thisSprite
repeat 100
  teach NN with input numbers from 1 to 4 and target output
  list 1 0 0 -1 by back-
  propagation with learning factor 0.1 on thisSprite
  NN show status with input numbers from 1 to 4 on thisSprite
```

1	0.7733923590532068
2	0.8594354585164047
3	0.7370385020970739
4	0.7172308713226372

length: 4

Training states after every 20 training steps:



The output of the net after training:

NN output of last layer with input numbers from 1 to 4 on thisSprite

1	0.7816390127483683
2	0.5277586070745496
3	0.5521002768260623
4	0.34392970102568626

length: 4

The positions of the largest or smallest value of the output can be determined directly using the *Math tools*:

maxpos of vector
NN output of last layer with input numbers from 1 to 4 on thisSprite



minpos of vector
NN output of last layer with input numbers from 1 to 4 on thisSprite

11.4 Traffic sign recognition with a neural network of perceptrons

"Deep" neural networks dominate the discussion about current "artificial intelligence". These are mostly "fully connected" networks consisting of several perceptron layers. "Fully connected" means that all neurons in one layer are connected to all of the next layer. Each connection is assigned a weight that indicates its influence on the connected perceptron - but you'd better read about that elsewhere.

Let's consider a network consisting of three layers, which receives as input the pixels of an actual 20 M pixel photo, i.e. 2×10^7 pixels. The input layer consists of $3 \times 2 \times 10^7$ MB numerical values between 0 and 255 (if we omit the transparency byte). To the next layer there are then $(6 \times 10^7)^2 = 3.6 \times 10^{15}$ connections - and then two more times. In total, $3 \times 3.6 \times 10^{15}$, i.e. about 10^{16} weights would have to be determined - a completely utopian task for "normal" computers. So, we will have to limit ourselves to somewhat smaller neural networks.

One way to train perceptron networks is to present them input vectors and the desired output at the same time. The network then computes the output resulting from the available, initially randomly chosen weights, and determines the difference to the given result. Starting from the last layer of results, it then "backpropagates" the weights so that its output fits the given result "slightly better". This procedure is called *backpropagation*. You should also read more about this elsewhere. The trained network results from many such corrections. "Learning" therefore means to adjust the parameters (the weights) based on many examples. With the help of these parameters, the net determines an output vector from the input vector: it calculates a function value. Our *NeuralNetPad* can simulate and train such perceptron networks.

The weights form a total *tensor* with m layers consisting of $n \times n$ matrices. *NeuralNetPads* should therefore master linear algebra. The only new feature is the *softmax* function  of vector  which can be used to scale input vectors, for example. You should also inform yourself about this.

The dimensioning and initial assignment of the network are done in the block *add new weights*. With this block we can create a new neural net of any size with random initial weights. In this case it has width 3 and depth 2.

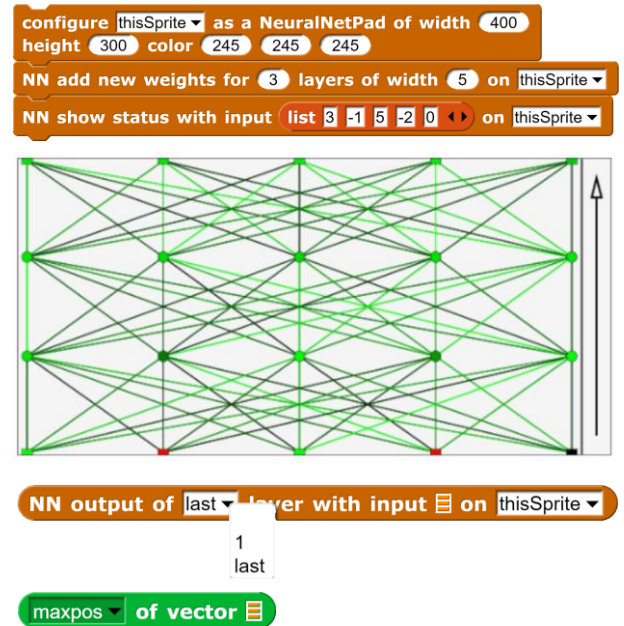
NN add new weights for 2 layers of width 3 on thisSprite

Since displaying the many numbers would be rather confusing and also hardly informative, the connecting lines (the edges) are color-coded according to the values of the associated weights: from full green for large positive values to black for small amounts to red negative weights. Since initially only positive numbers are drawn at random, a new net is predominantly green. It shows what results from the calculations with the input vector to be specified.

NN show status with input  on thisSprite

The vertices of the net are color coded like the edges. At the bottom, the elements of the input vector are shown as small rectangles. The inner layers form colored circles and the last layer is again shown as an output layer in rectangular form. The direction of the calculation from bottom to top is shown by the arrow on the far right. Since you can easily rotate sprites, the direction can of course also be displayed differently.

A simple initial configuration would thus result in:



Often you need the results of the last or an inner layer of the net. These can be calculated with the help of the *output of...* block for a given input. Since the color coding does not necessarily show the largest or smallest element clearly, this can be determined with the help of the *...of vector...* block.

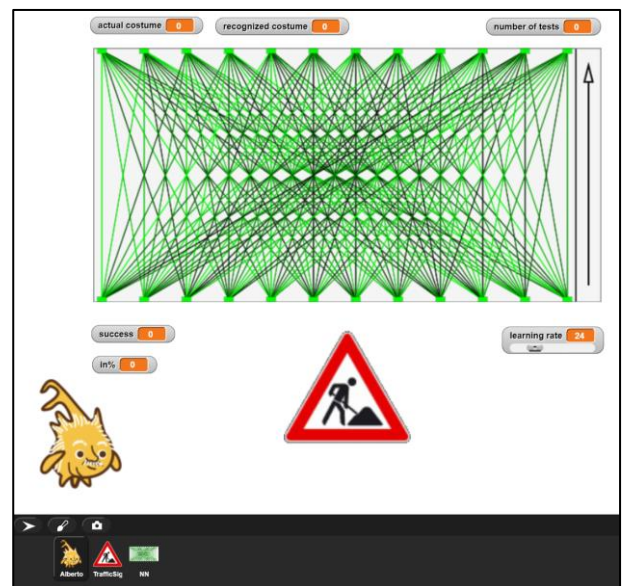
The training of the net is done with the help of the block *teach NN...* by backpropagation with a learning factor to be specified. The learning factor may be somewhat larger at the beginning, and then be reduced.



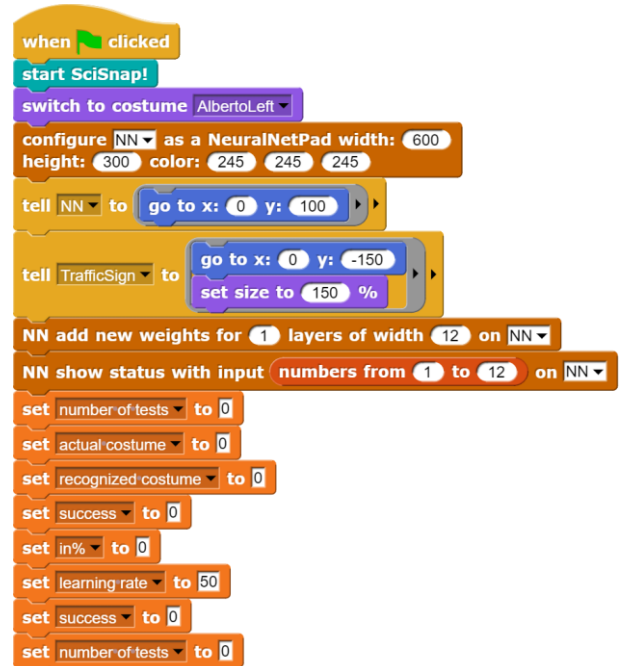
We want to train a neural network (NN) to recognize 12 different traffic signs. To do this, we search for images of these traffic signs in the network and reduce them to the format 100 x 100 pixels. Now they can be displayed well on the screen, but the 10,000 pixels are of course much too much as inputs for a NN.

To bring the amount of data within tolerable limits, we reduce the pixels to a 2x2 format by *mean-pooling*, i.e. we average the color pixels in each of the four quadrants of the image. The 30,000 values of the traffic sign image thus become 12.

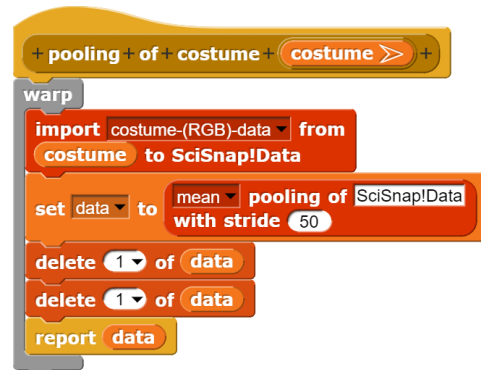
Because it is a difficult problem, this time *Alberto* takes the overall control.



To start, *Alberto* gives the NN sprite a new costume. Then he creates the weights for a new (here: 12x1) net in the NN. This he lets draw with a (still nonsensical) input. Then he sends the NN to a well chosen place in the upper center and does the same with the traffic sign below. Finally some variables are set to 0. We will need them later.



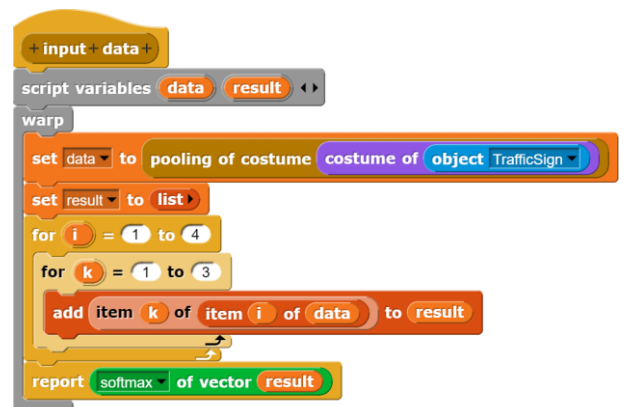
Alberto first needs to reduce the amount of image values using the *pooling* operation. To apply the operation, he must import the image data. Then he can convert them, delete the dimensions of the reduced image specified at the front of the list, and return the result. We summarize everything in a new block *pooling of <costume>*.



4	A	B	C	D
1	176.2016	61.1332	59.3068	200.328
2	176.0848	59.7848	59.4004	200.43
3	171.4432	53.4512	53.4504	199.512
4	169.4336	43.096	45.4232	199.41



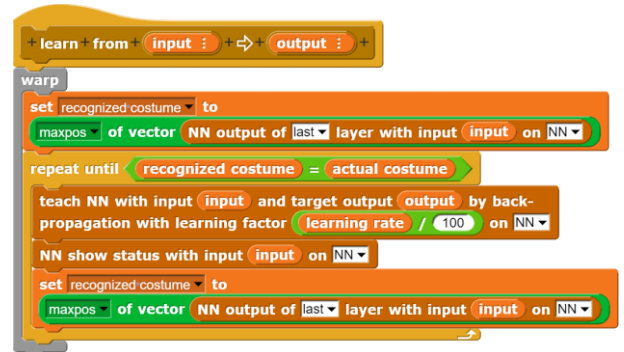
The color values of the reduced image are assembled by *Alberto* into an input vector. These are modeled with the softmax function of the *Data tools* in order to exclude unfavorable input values.



Accordingly, the training vector in *training data* can be determined with the searched output values of the NN. In our case, all values should be 0 except for the one corresponding to the costume number of the traffic sign.



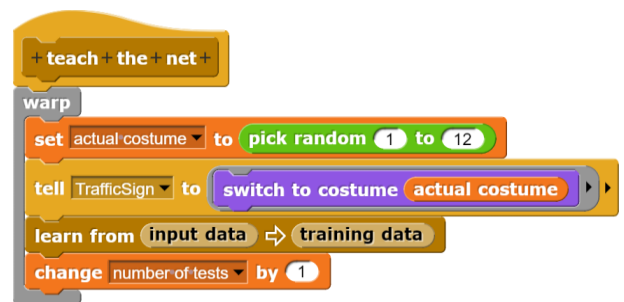
The NN receives two new methods *learn from...* and *test with...* When learning, the position of the place with the largest output value of the NN is determined and compared with the current costume number of the traffic sign. If these values do not match, then learning continues.



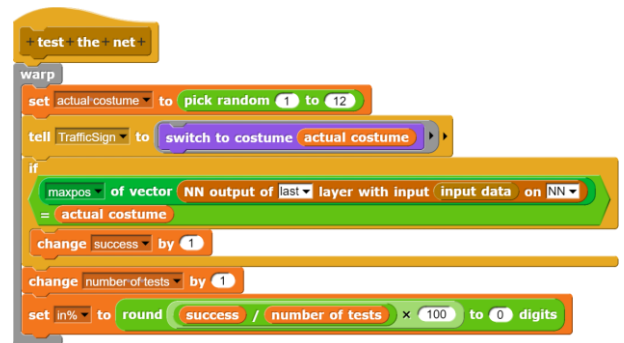
For testing, the same operation is performed only once.

Now we have everything together to let *Alberto* work reasonably.

A teaching operation consists of determining a random costume number with the corresponding costume change. Then the learning process of the NN is started with new input and target data. The passes are counted.



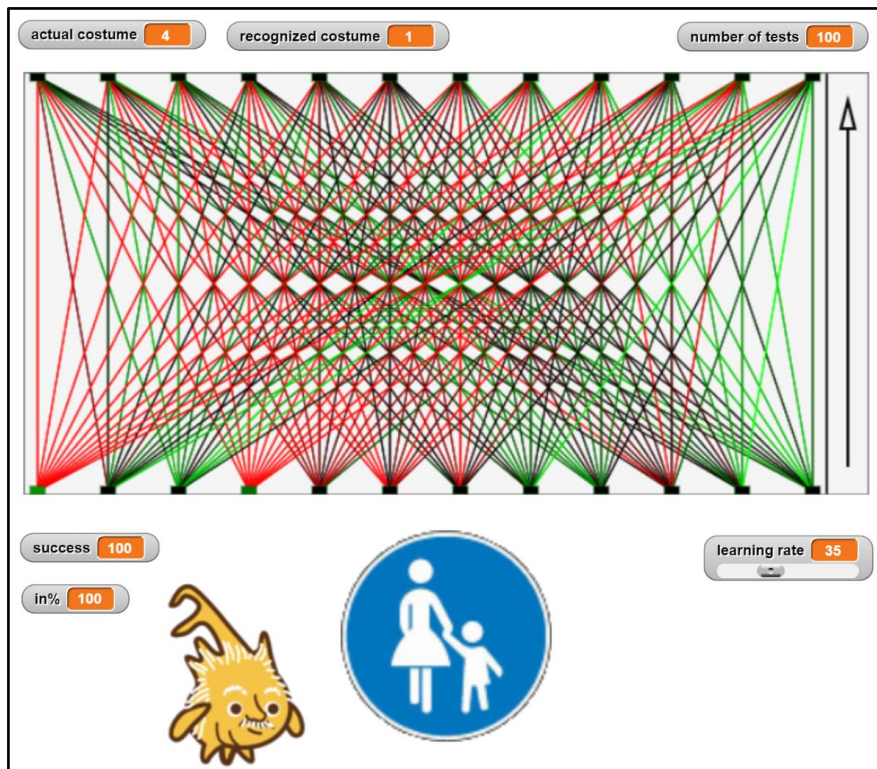
For testing, a similar procedure is followed: The costume is changed and it is checked whether the NN calculates the correct costume number. If this is the case, then everyone is happy. The percentage of correct attempts is determined.



Multiple learning and test runs can then be easily triggered.



After several hundred training runs with a higher learning rate and again with a lower one for fine tuning, we achieve recognition rates of up to 100%.



Tasks:

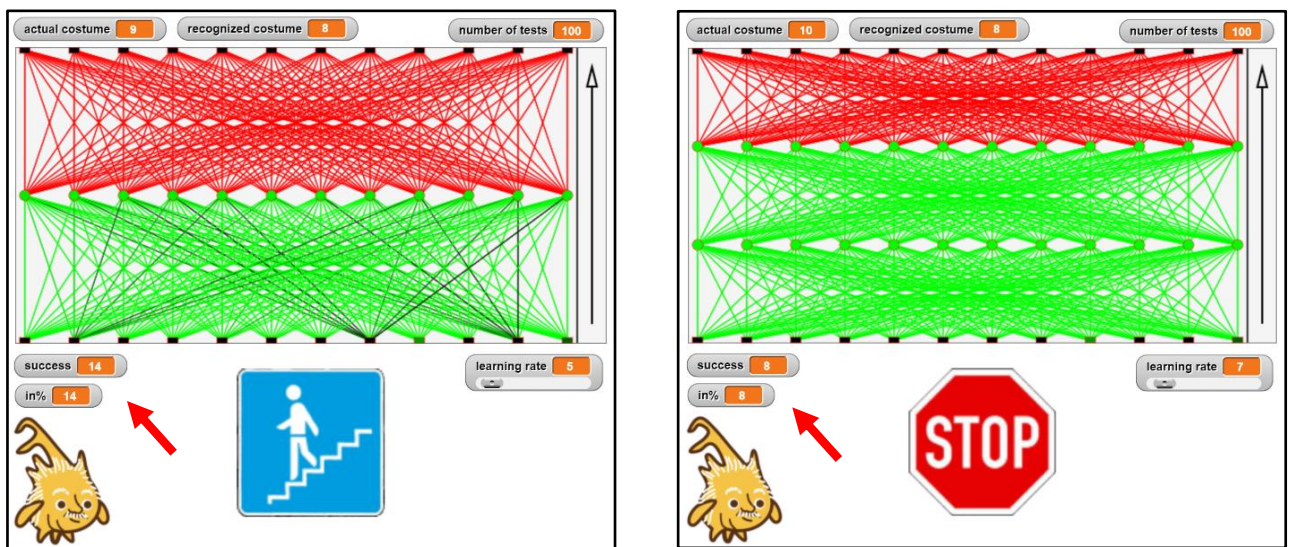
1. Train a single-layer network with different learning rates and numbers of learning passes. Determine the recognition rate as a percentage in each case.
2. Plot the results from 1. graphically using a PlotPad.
3. Experiment with multi-layer NNs. Will the results be better?
4. Increase the length of the input vector by changing pooling. Will the results be better?
5. Increase the number of recognizable signs by allowing more than one 1 in the output.

11.5 Under- and Overfitting

Machine learning uses training data to adjust the parameters of a function so that other values are predicted well - if everything works well. So, you build a prediction tool, a kind of "telescope" for data.

One might think that the more customizable parameters a function contains, the better it is. But this is not the case. On the one hand, (1.) more parameters require more training data and training runs, i.e. more learning time; on the other hand, (2.) an "unsuitable" number of parameters can also prevent "good" solutions. For both we give an example.

Re 1: In the neural network for traffic sign recognition (last example), we achieve very good results with one layer. If we increase the number of layers and leave the number of training runs the same, then the recognition rate decreases drastically.



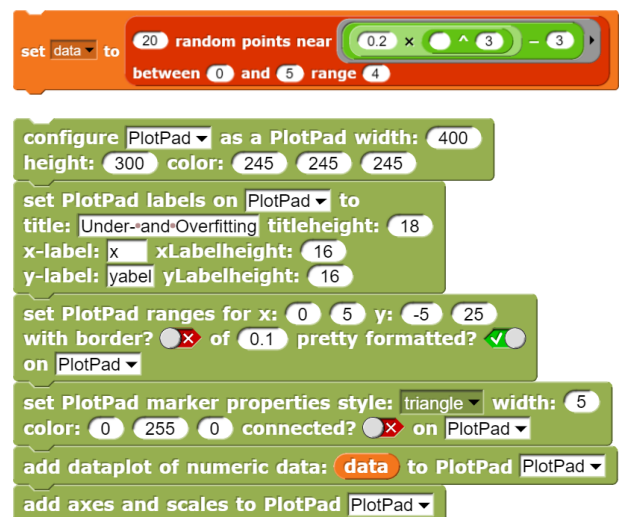
Re 2: If the training data is well reproduced by the function, it does not mean that this is also true for other data. It depends very much on the kind of function that is generated. As application we choose the example *polynomial interpolation*.

The task is: *With the help of training data, the coefficients of a polynomial are adjusted so that OTHER data are predicted as well as possible.*

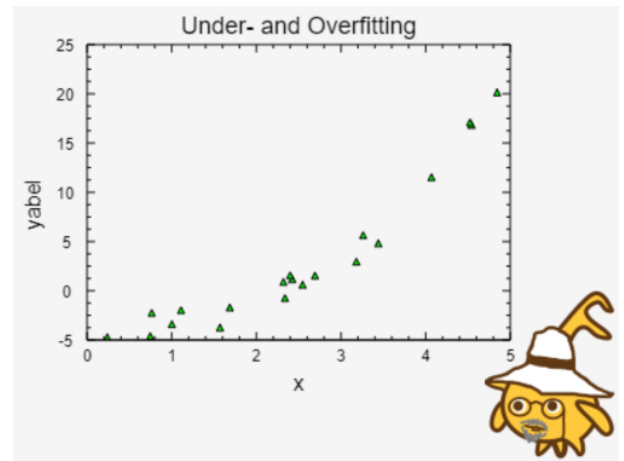
To do this, we need to generate data that will be used to calculate an interpolation polynomial. This time the task is done by *Hilberto*. He generates data that scatter around the parabola $0,5 * x^2 - 3$.

We configure a second sprite next to *Hilberto* named *PlotPad* as *PlotPad* and plot the data on it.

As a "workhorse" we choose the *PlotPad*. If necessary, we import the required functionality from other libraries.



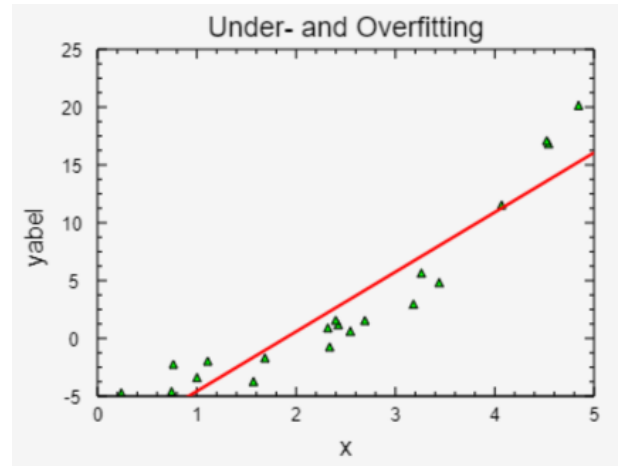
This is already enough to show the data.



We first try the interpolation with a regression line.

```
set PlotPad line properties style: continuous
width: 2 color: 255 0 0 on PlotPad
add graph regression line parameters of data to PlotPad PlotPad
```

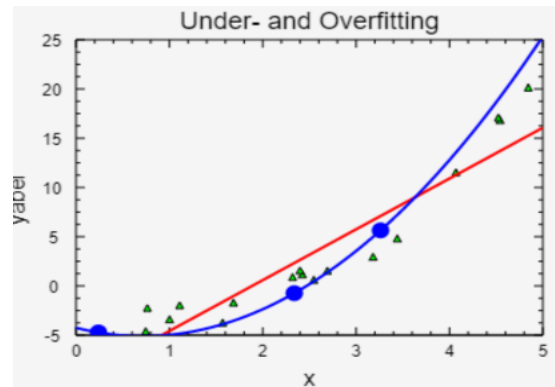
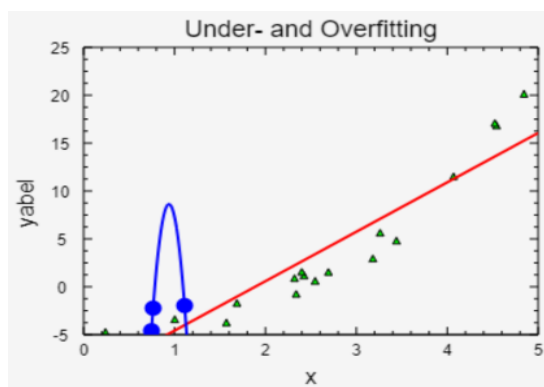
This actually looks quite nice, but it doesn't properly fit on the sides.



So, we try it with a polynomial interpolation.

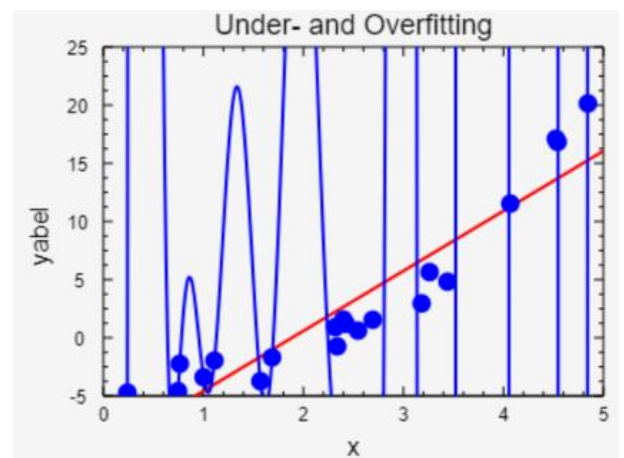
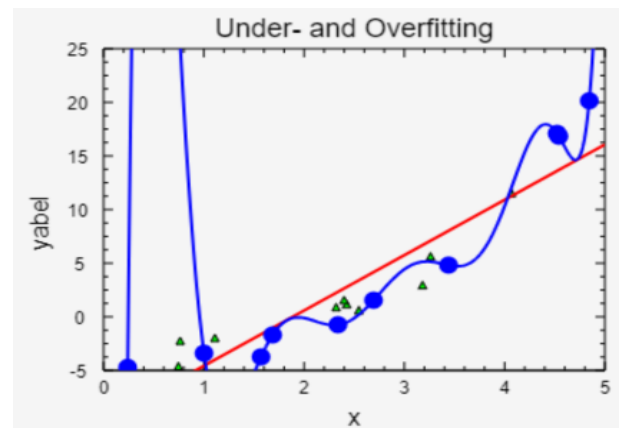
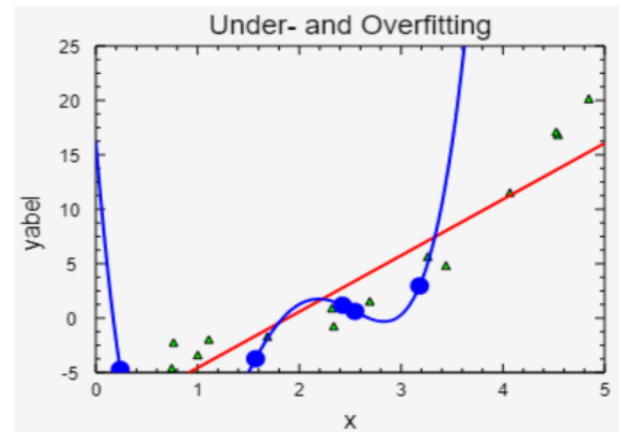
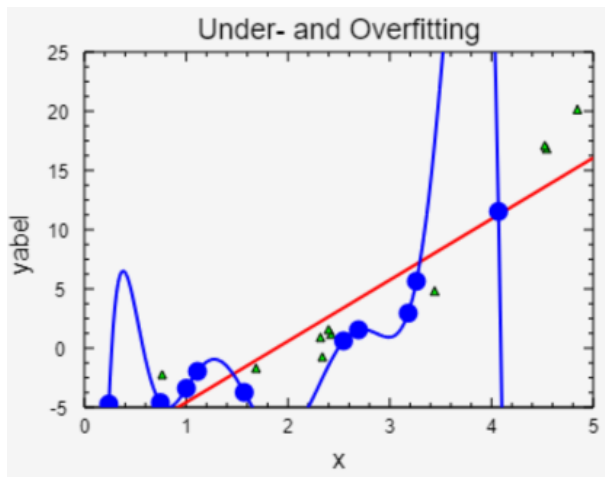
First of all, we choose three random pairs from the training data, determine the interpolation polynomial and draw it. Because we want to experiment further, we generalize the solution to a polynomial through n points. The results depend on which points were caught. Here is a bad and a quite good result.

```
interpolation polynomial for n # = 3 + random points in data
script variables points
if n < 2 or n > length of data
  say ERROR: impossible for 2 secs
  stop this script
set points to list
repeat until length of points = n
  add item pick random 1 to length of data of data to points
set points to points without duplicates
set PlotPad line properties style: continuous
width: 2 color: 0 0 255 on PlotPad
set PlotPad marker properties style: o circle width: 10
color: 0 0 255 connected? x on PlotPad
add dataplot of numeric data: points to PlotPad PlotPad
add graph polynomial interpolation for points points to PlotPad PlotPad
```



Now we're getting brave! Instead of three points, we choose 5. After all, we want to do a good job! That works halfway in the middle, and then - oops!

Maybe we just have to use more points. Let's try 10. The polynomials run through more points, but they "take off" at the edges.



Well, then with all the points!

You can see that as the degree of the polynomial increases, there is more training data directly on the graph, but that in between, the wild oscillations of the polynomial only "predict" nonsense values.

The quality of what we learn therefore depends very much on how we deal with deviations. We have to decide which inaccuracies in detail are tolerable so that the forecast as a whole becomes reliable. If the degree of the polynomial is too small, we speak of *underfitting*, if it is too high, of *overfitting*.

Tasks:

1. Discuss different ways of defining a "good" degree of the interpolation polynomial (i.e., its highest power).
2. Formulate your results so precisely that they can be realized as scripts.
3. Test the scripts on different data sets.

11.6 Classification in the HR-diagram according to the kNN method

In the Hertzsprung-Russell-Diagram (see Wikipedia) the luminosity of stars is plotted above their stellar class. The result is a kind of line from left-top to right-bottom, the "main sequence". On this line stars like the Sun are predominant. Right-top above the main row we find the red giants, left-bottom below the main row the white dwarfs. That's enough for now. (Image source: [HR])

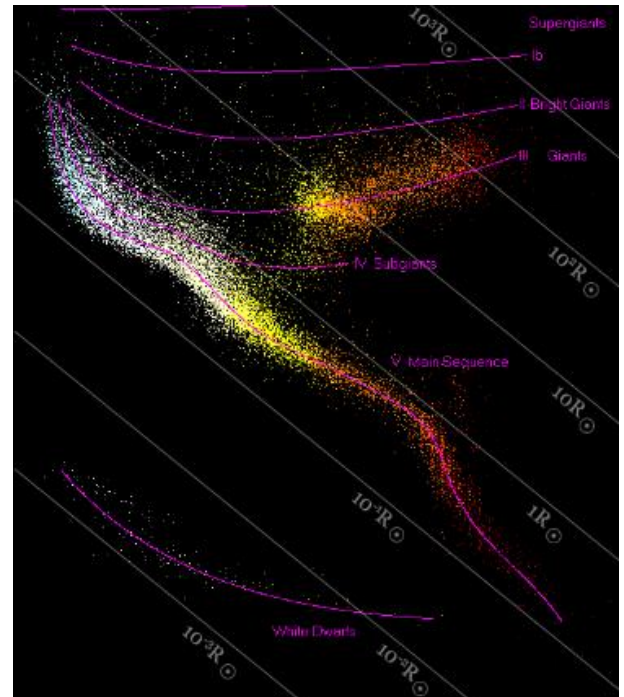
We want to classify new stars in this diagram using the *k-nearest neighbor (kNN)* method: We generate as training data a list of stars with their coordinates (simply as image coordinates in the diagram) and their type. If we want to classify a new star, we determine its position in the diagram and find the nearest k (e.g. $k=5$) neighbors. Then we determine the most frequently appearing star type in this list. We assign it to the new star.

First of all, we need an image of the Hertzsprung-Russell-Diagram ([HR]). We import this into *Snap!* as a costume of an *ImagePad* and generate the required data from it. Since we want to draw on the image, we work with a copy of the HR diagram in order not to change the original.

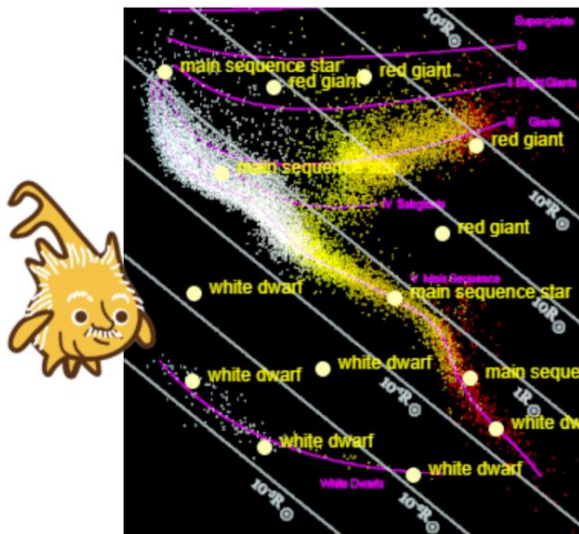
We obtain the training data by specifying a star type and then clicking on some points in the diagram that correspond to this type.

After that we can classify new stars by clicking and labeling them.

We set some properties for the display and draw a circle at the location of the star. Then we determine the five nearest neighbors and the number of occurrences of their type. As a result, we delete the headings and sort the list in descending order. The type of the new star is the first element in the first line. We write this next to the star.



The result:



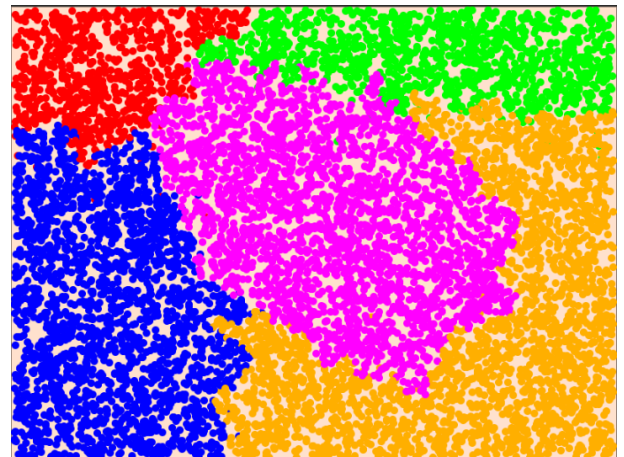
```

+ draw + type + of + star + point : +
script variables neighbours startypes type
set ImagePad line properties style: continuous
width: 1 color: 255 255 0
fill color: 255 255 180 on thisSprite
fill circle center: item 1 of point item 2 of point radius: 5 on
thisSprite
set neighbours to 5 next neighbors of point
in starData
set startypes to number of column 4 of neighbours
grouped by column 4 considering headline?
delete 1 of startypes
set startypes to startypes sorted by column 2
ascending considering headline?
set type to item 1 of item 1 of startypes
draw text type at 10 + item 1 of point item 2 of point
height: 12
horizontal? on thisSprite

```

Tasks:

1. Add the newly classified stars to the sample list so that they are included in further classifications.
2. Draw different colored dots in the right places on the sprite for the different types of stars, instead of labeling them.
3. Run the process for randomly selected points. Does the same pattern always emerge? Do completely different or similar patterns emerge? What does it depend on?



11.7 Decision trees according to the ID3 method

A decision tree reproduces the process by which "experts" arrive at a decision. A simple example is the children's game "Guessing animals", where questions are asked one after the other ("Does it have four legs?", "Does it have fur?", "Is it the Easter bunny?", ...), which exclude more and more animals until either one remains or there is a dispute. In our case, the expert knowledge manifests itself in the form of classified data, where the classification should be in the last column - for simplicity as a binary decision. From these data sets, a decision tree is constructed, which other, not yet "labeled" data can traverse in order to be classified. One method for constructing such trees is the ID3 method (Iterative Dichotomiser 3)²⁷. It creates wide trees that can be traversed quickly by calculating the entropy of the individual attributes.

A decision tree consists of nodes and edges that start from a *root*. The nodes connected layer by layer by edges are either *inner nodes*, from which still further edges start, or *leaves*, which mark the end of a branch. In the tree construction it must be decided in which order the attributes are used, which promises the largest information gain at a certain point. Usually, the information gain is calculated with the help of the *entropy*, which gives information about the "disorder" of a system. If we know the proportion p of an attribute value, then the entropy S is defined as $S(p) = -p \cdot \ln p$. For an attribute A with n attribute values with proportions p_i , the entropy is then the sum of the partial entropies: $S(A) = \sum_{i=1}^n S(p_i) = -\sum_{i=1}^n p_i \cdot \ln p_i$. The *information gain* that comes from deciding on an attribute value is given by the weighted entropy $gain \approx \sum_{i=1}^n \frac{n_i}{n} \cdot p_i \cdot \ln p_i$. You should look elsewhere for the details of the procedure!

To construct and query decision trees, *SciSnap!* has three blocks in the *Data tools*. The entropy of a list is determined by the block **entropy of**, **decision tree ID3 for** with labeled data in last column constructs the tree, and the tree is queried with **classify** with ID3-tree.

As an example, let's explore whether there are "secret" relationships hidden from our untrained eyes in the NYCitibike data you've already encountered. For example, it could be that the borrowing data can be used to infer the gender of the borrowers. Let's try it.

We load the full dataset with 577703 elements, which we "smooth" somewhat by omitting the location data (*latitude* and *longitude*), because the borrowing stations result from the other data. We shorten the time scales to the hour of the day, as in the other examples, and we delete the unlabeled data (*gender unknown* : "0").



²⁷ J.R. Quinlan, 1986

For training, use an abbreviated dataset with 1,000 borrowing records, i.e. to construct the decision tree. For this we simply pull 1,000 records from the total dataset.

Then the ID3 tree is created from the data. This takes some time, here: 35.2 seconds.

After that we test the tree with randomly selected data from the training dataset. This should do the job.



test ID3-tree decisionTree with tests 100 of dataset training set

100%

It does.

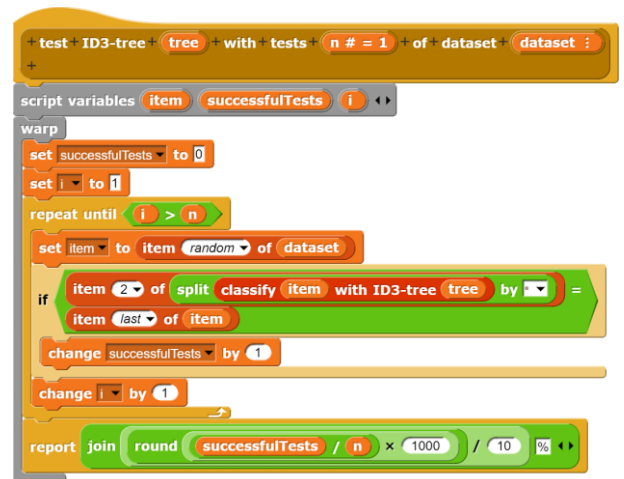
Now we use the full data set with all records, of which 336,955 are still left after filtering. Maybe this shows the incredible power of "artificial intelligence"??

test ID3-tree decisionTree with tests 10000 of dataset data

12%

Well. By pure guessing we would have been much better! Either the AI we created, the decision tree, has no uncanny power at all, or the data were unsuitable for the question, or we are dealing with a typical overfitting, because the training data are perfectly identified, but the rest almost not at all. Look for the reason!

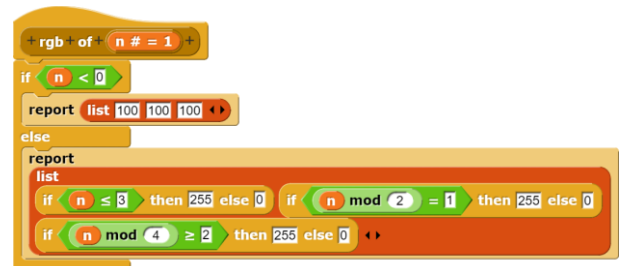
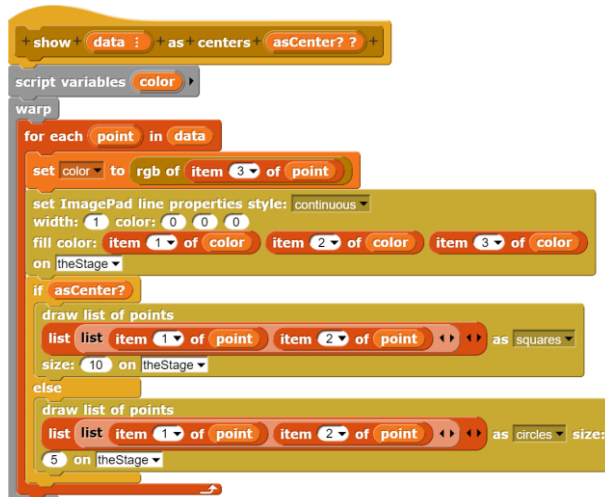
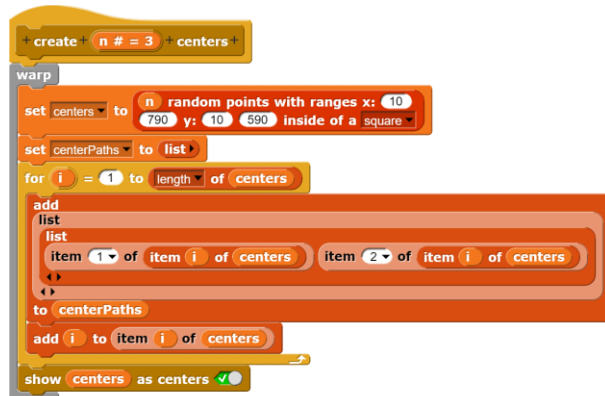
At least we learned how to construct and use a decision tree. One must be grateful! 😊



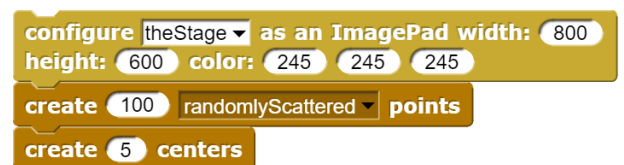
11.8 k-means-Clustering

The *Data tools* palette provides blocks for *k-means clustering*, but they (of course) only provide the final result. The method is used to distribute k centers among n data points in such a way that the resulting k groups are as evenly distributed as possible and the distances to the centers are minimal. An example may be a housing estate in which k distribution centers are to be established, e.g., for telephone connections or electrical energy. In this case the Euclidean distance will be taken, but other metrics are possible, e.g. along existing roads or the Levenshtein distance.

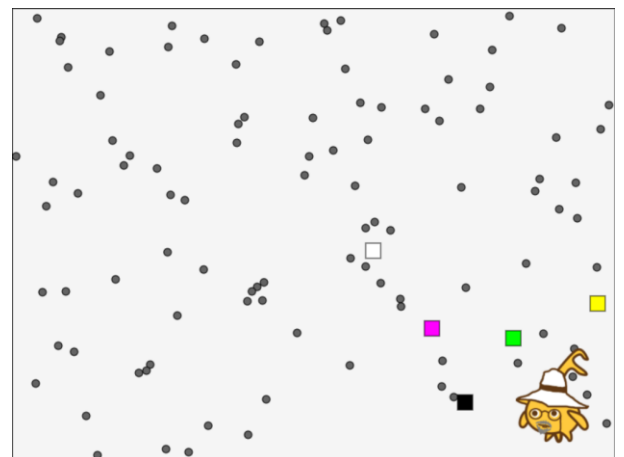
We want to illustrate the entire process here. To do this, we create a set of, for example, 100 random points with "JavaScript coordinates" to which we append the cluster number. This is initially "-1", because no clustering has taken place yet. For this we create " k ", e.g. 5, "centers". Points and centers are displayed as circles or squares on the stage. The colors, initially gray, determine the cluster numbers.



The sequence of commands

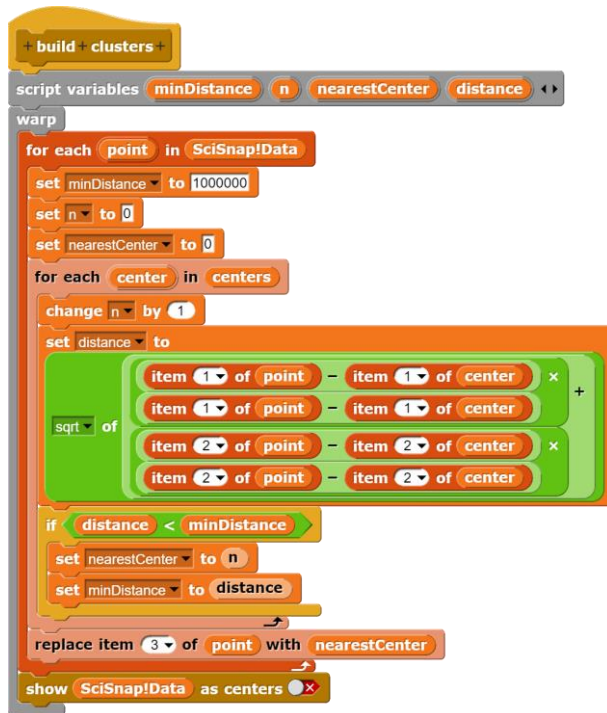


results in the following picture, for example.

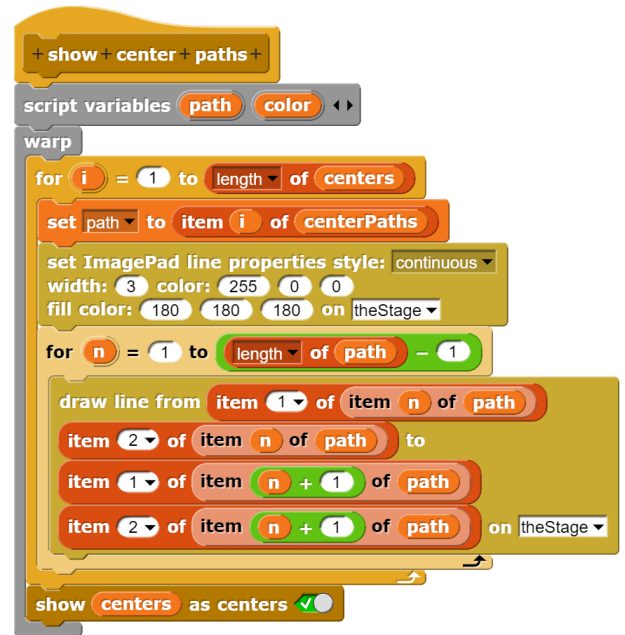
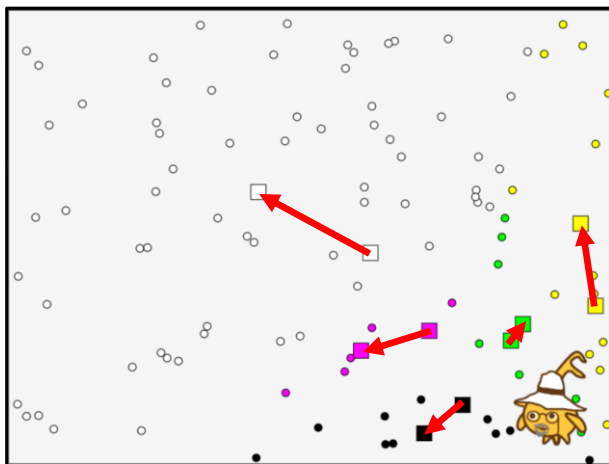
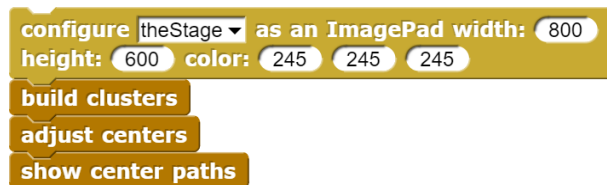


We number the centers and enter their coordinates in a list *centerPaths* so that we can track their movements.

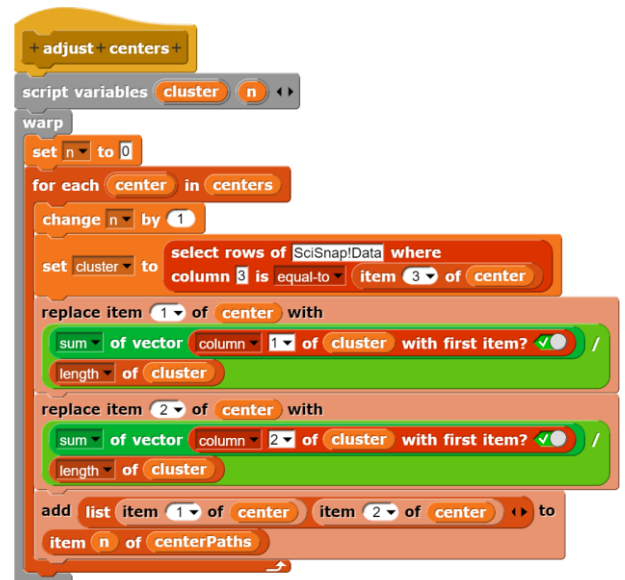
The k-means method now determines the nearest center for each point and colors the points in its color.



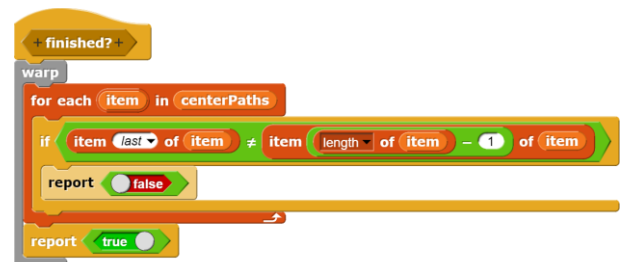
In summary we get:



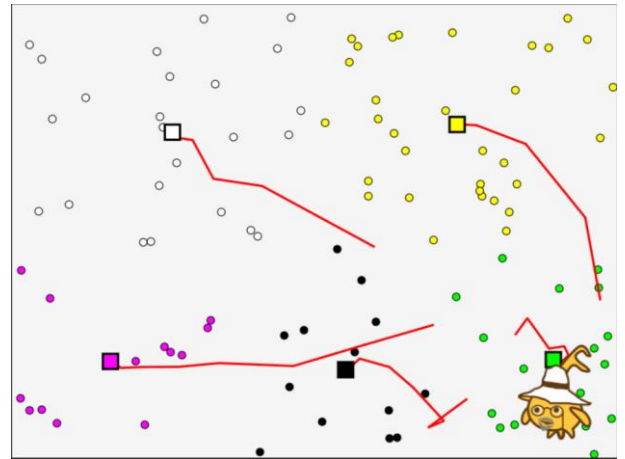
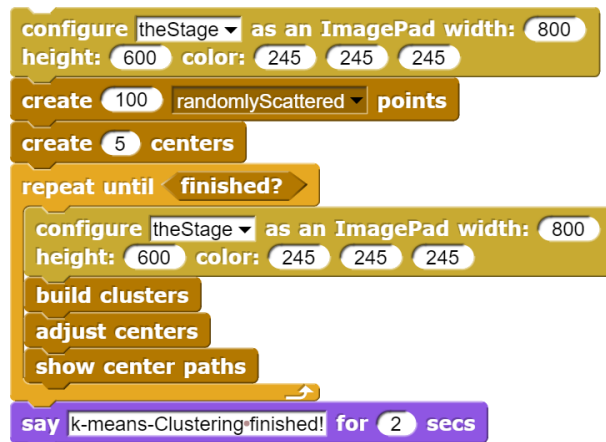
Now the centers are moved to the center of the set of points assigned to them. The new positions are entered in the position list.



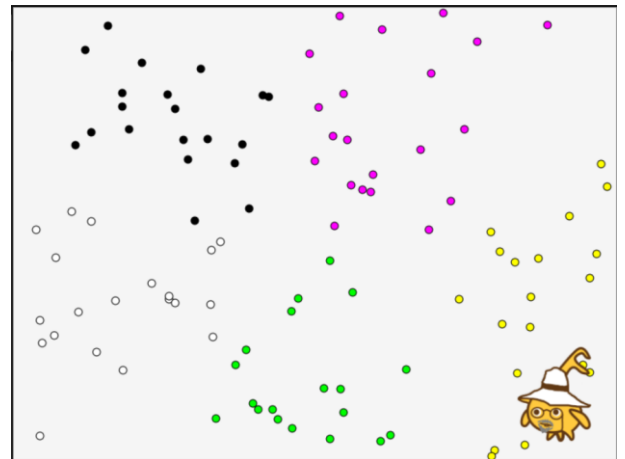
In most cases, of course, we have not yet achieved an even distribution among the centers. Therefore, the procedure is continued until there are no more shifts in the centers..



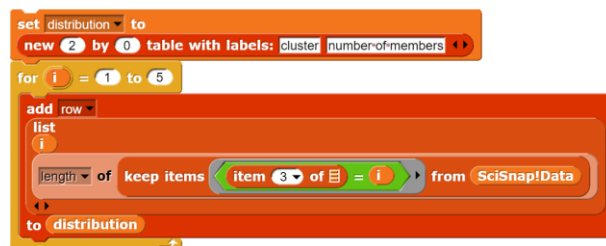
We get the result:



Using the existing blocks, we could of course have obtained the result a bit simpler - but just without representing the process:



Let's take a quick look at the distribution of points among the five clusters:



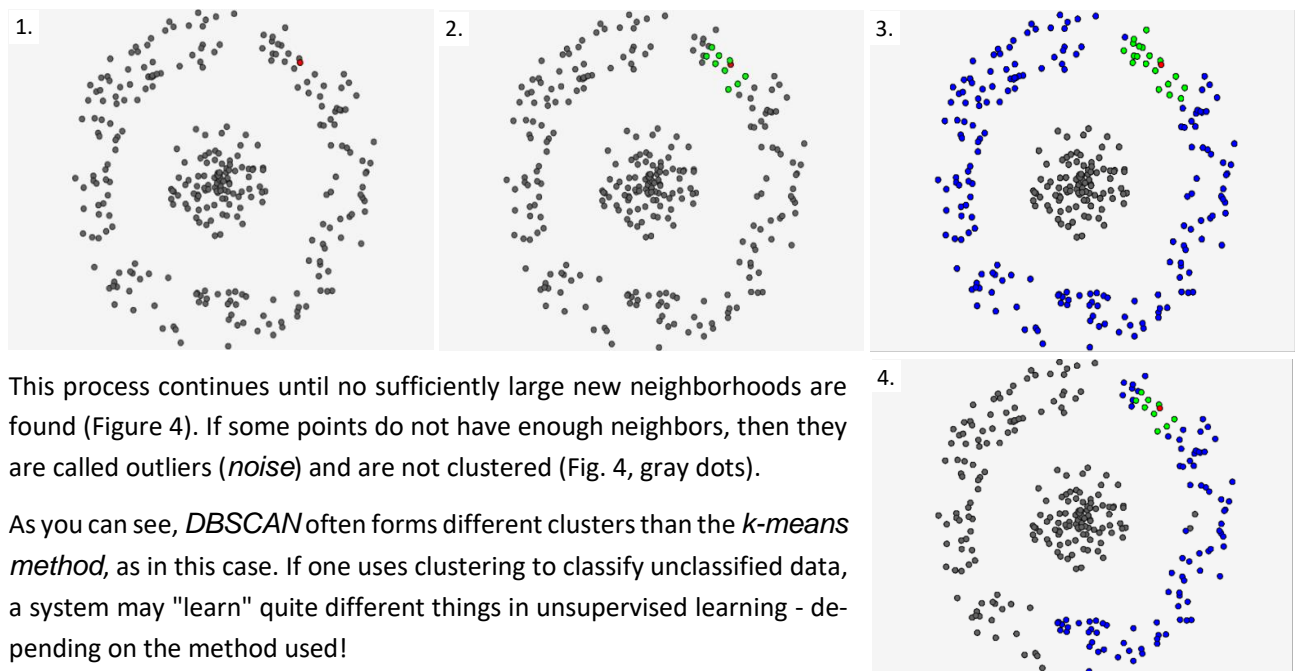
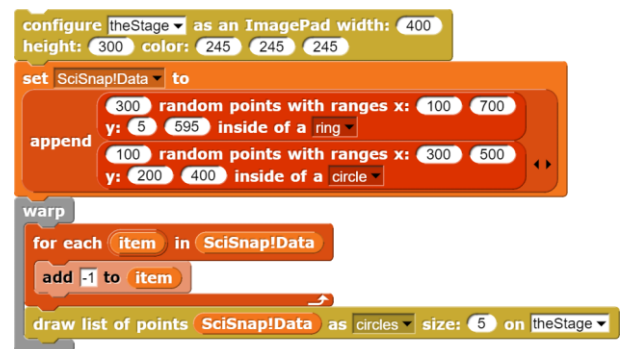
The result is encouraging. If it were a matter of distributing telephone switchboards, then the construction crews could come now. 😊

distribution		
6	A	B
1	cluster	number of members
2	1	20
3	2	21
4	3	20
5	4	20
6	5	19

11.9 Clustering according to the DBSCAN method

The *DBSCAN* (*density based spatial clustering of applications with noise*) method tries to identify "density islands" in the data space. Before starting the procedure, two parameters must be selected: a *radius* specifies up to which distance points are to be considered "neighboring", and a *minimum group size* specifies from which number on a group of points is to be considered. The process goes through all data points and determines their neighboring points with the help of the radius. If enough are available, then all found neighbors are examined again for neighborhoods. If these are large enough, they are added to the original one. In this way, a coherent "point cloud" of related points is created - a cluster. If not all data points have been recorded, the next unprocessed data point is added and a new cluster number is assigned.

We create a point distribution and select any point from it. We count its neighbors within the radius (Figure 2) and get 5 neighbors. If this is enough, we add their neighborhoods to the original one - if they are large enough (Fig. 3)..



This process continues until no sufficiently large new neighborhoods are found (Figure 4). If some points do not have enough neighbors, then they are called outliers (*noise*) and are not clustered (Fig. 4, gray dots).

As you can see, *DBSCAN* often forms different clusters than the *k-means method*, as in this case. If one uses clustering to classify unclassified data, a system may "learn" quite different things in unsupervised learning - depending on the method used!

For somewhat larger data sets, it is useful to use the *DBSCAN clustering for ...* block available in *SciSnap!* which works according to the following procedure.

DBSCAN clustering for SciSnap!Data **radius** 50 **minMembers** 5

So let's implement the procedure.

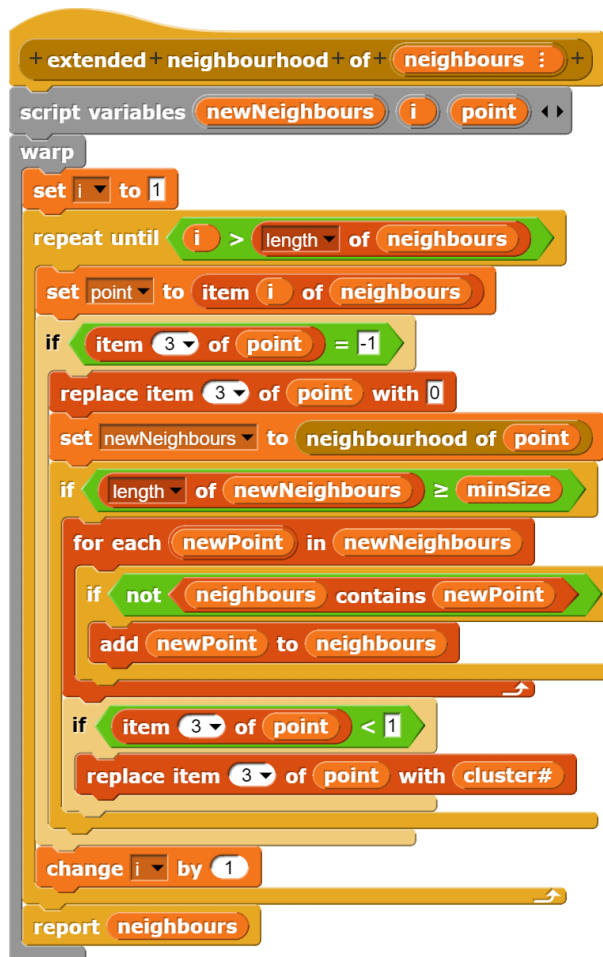
First, the set of points is created and assigned to the *SciSnap!Data* variable. The points are drawn.

Then we set some initial values and draw the starting point.

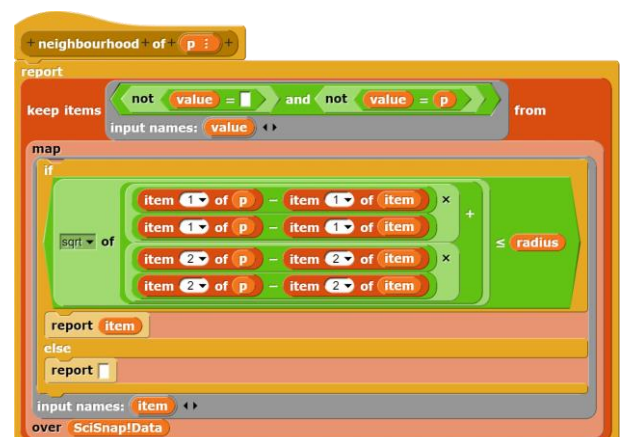
The first neighborhood is calculated and displayed.

After that, the DBSCAN procedure starts: For all points in the neighborhood, their neighborhoods are determined, checked and, if necessary, transferred to the old neighborhood. If necessary, the generation of the next cluster is started.

The two additional blocks used are ...



... and



11.10 Outlier detection according to the DBSCAN method

In many cases, it is not the clusters that are interesting, but rather the outliers, i.e. data that "do not fit the grid", that are conspicuous. The *DBSCAN* method is well suited for this, because it does not cluster "outlying" points (*outlier detection*).

We create suitable random data, i.e. two "heaps" and some points scattered over the stage, some of which are on the outside. We cluster this data using the *DBSCAN* block.

397	635.999983	528.288293	2
398	606.007918	513.330165	2
399	558.459258	512.274097	2
400	656.989110	6590.342389	2
401	780	537	-2
402	79	527	-2
403	643	243	-2
404	519	245	-2
405	61	201	1
406	426	325	-2
407	98	204	1
408	591	347	-2
409	165	304	1
410	765	207	2

DBSCAN clustering for SciSnap!Data radius 50 minMembers 5

From this list, we pick out the outliers as non-clustered "noise" and display the points in red. We combine the blocks for this purpose.

show keep items item 3 of 0 from DBSCAN clustering for SciSnap!Data radius 50 minMembers 5 in 255 0 0

The example is of course unrealistic in its shortness. But if we look at parameters of e-mails such as the structure of the header, number of addressees, type of attachments, keywords, ..., then we come into the realm of real applications, such as spam detection.

Two things follow from these considerations:

- On the one hand, the data obviously have to be processed heavily in order to be represented in a suitable high-dimensional space. Procedures must be developed to assign a vector to the data points - and vice versa. The actual clustering thus is no longer the focus at all: it is only a step within the process.
- On the other hand, the number of parameters increases the number of dimensions of the data space, so that the available data hardly have any neighbors around them. This *curse of dimensionality* can only be countered if an extremely large amount of data is available. We have another example of how machine learning is not only about the technology of the computers, but also about the availability of data.

configure theStage as an ImagePad width: 400 height: 300 color: 245 245 245

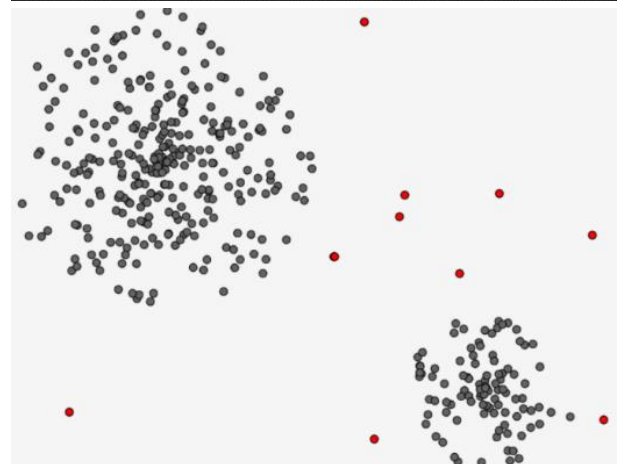
set SciSnap!Data to

300 random points with ranges x: 5 400 y: 5 400 inside of a circle

append 100 random points with ranges x: 450 795 y: 400 600 inside of a circle

20 random points with ranges x: 5 795 y: 5 595 inside of a square

show SciSnap!Data in 100 100 100



11.11 DNA-Clustering with Levenshtein distance

DNA studies often rely on dissecting the DNA, examining DNA snippets, and reassembling the DNA if necessary. In this case, we want to dissect a "soup" with different DNA snippets into three similar groups, where the distance of the DNA sequences is measured using the Levenshtein distance for strings.

DNAs consist of sequences of bases A, C, G and T (see literature). Since our blocks work with averaging and thus with numbers, we map such sequences to an n-dimensional space whose coordinates take only the values 1 to 4. So we have to map DNAs to such vectors, process them and transform back.

We start by generating the DNA snippets. We limit them all to the length of 20 and form three clusters using the Levenshtein metric.

After that, the three clusters are extracted and the cluster number is detached.

Then, the vectors of the clusters can be translated back into base sequences.

For testing, we form the mean Levenshtein distances within a cluster and between two clusters.

```

+ about + n # = 100 + DNA-snippets + with + length + laenge # = 20 +
script variables DNA result zz
warp
set result to list
repeat n
  set DNA to 
  repeat laenge
    set zz to pick random 1 to 4
    set DNA to
    if zz = 1 then join DNA A else
    if zz = 2 then join DNA C else
    if zz = 3 then join DNA G else join DNA T
  add DNA to result
report result without duplicates

```

```

start SciSnap!
set SciSnap!Data to about 100 DNA-snippets with length 20
set clusteredData to
  3 -means clustering for map DNA → vector over SciSnap!Data
  with metric
  Levenshtein-distance of Vektor → DNA and Vektor → DNA
set cluster1 to select rows of clusteredData where
  column last is equal-to 1
set cluster2 to select rows of clusteredData where
  column last is equal-to 2
set cluster3 to select rows of clusteredData where
  column last is equal-to 3
delete column last of cluster1
delete column last of cluster2
delete column last of cluster3
set cluster1 to map Vektor → DNA over cluster1
set cluster2 to map Vektor → DNA over cluster2
set cluster3 to map Vektor → DNA over cluster3

```

```

mean of vector
map
  mean of vector
  map
    Levenshtein-distance of DNA1 and DNA2 input names: DNA2
  over cluster3
input names: DNA1
over cluster3

```

10.979206049149338

```

mean of vector
map
  mean of vector
  map
    Levenshtein-distance of DNA1 and DNA2 input names: DNA2
  over cluster3
input names: DNA1
over cluster2

```

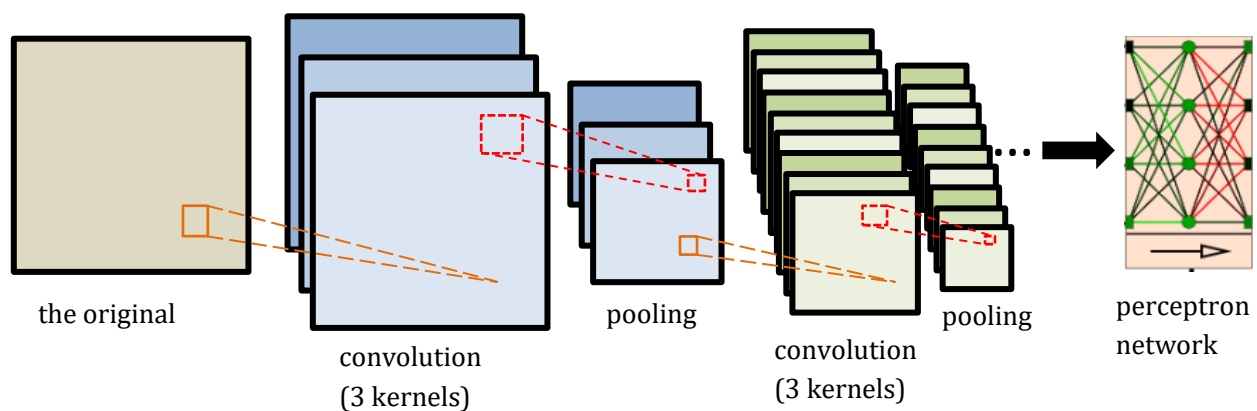
11.803689064558629

All right - at least they differ a little.

11.12 Character recognition with a Convolutional Neural Network

The immense number of parameters in fully connected perceptron networks and the consequent need for huge amounts of training data has led to other network variants to drastically reduce this number. One of these is *Convolutional Neural Networks (CNNs)*, in which the amount of input data to the perceptron network is reduced. This type of network is used very successfully in image and speech recognition, for example.

CNNs reduce the amount of data by first applying several *kernels* in a multi-step process that filters out certain properties of e.g. an image (edges, oval surfaces, ...) and thus results in several *feature maps* that usually have the same size as the original. This first increases the amount of data. Afterwards, a nonlinear *activation function (ReLU)* is usually applied to the feature maps, followed by a *pooling operation* that reduces the amount of data again. Mostly this is max-pooling, where the maximum value is determined from a section of the data. If you do this with a "window" that is moved over the feature map with a certain step size (*stride*), then each pooling step generates a value of the next, reduced feature map.

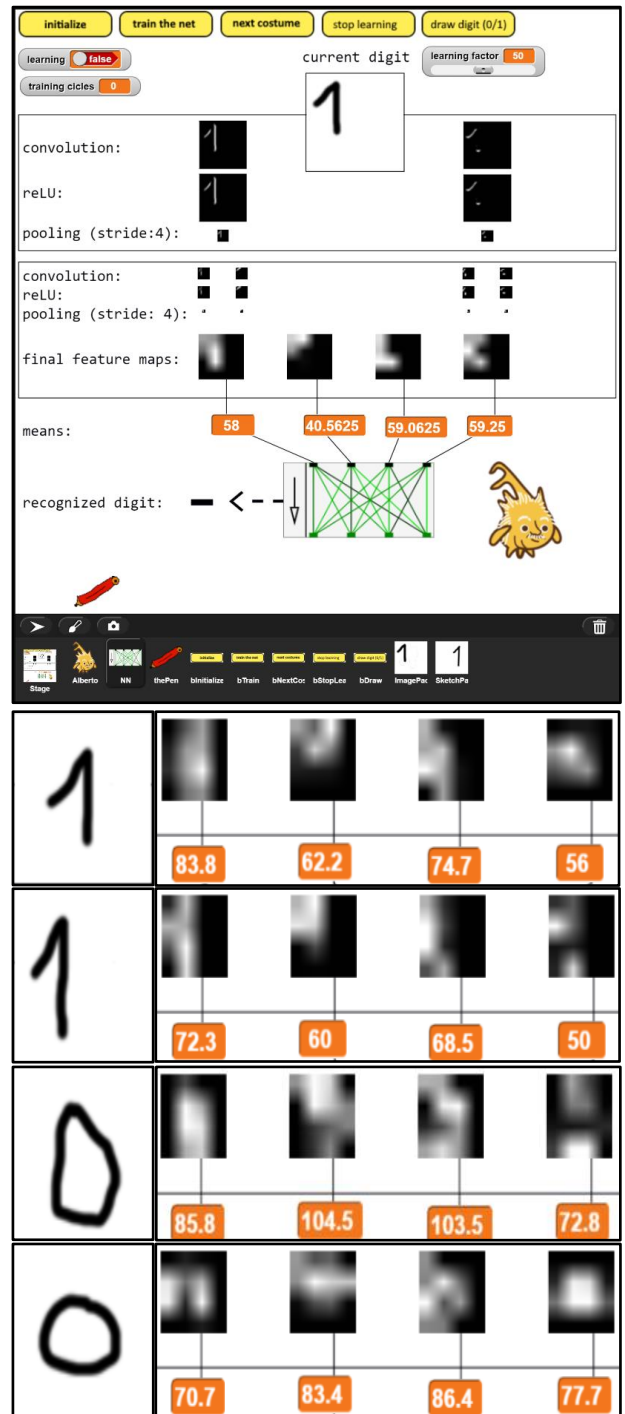


As an example, let's take a kernel that filters vertical lines: it colors a point white if there is a second pixel next to the point, otherwise black. In the "folded" image we can then recognize vertical lines of the original as bright spots. If it does not matter so much where exactly these lines are, then we do not lose too much information in pooling. A white spot in a feature map after various pooling processes then means: "*There was a vertical line somewhere in this area.*" Based on such data from several feature maps, it can then be deduced, for example, that a horizontal line, i.e. a corner, was also located there. If we had searched for "*beige*" areas as well as "*oval*" shapes, then the chance of identifying faces would not be so bad at all.

We now want to build a model for such a CNN that can distinguish the handwritten digits zero and one. For this we use the *Data tools* for auxiliary operations, an *ImagePad* for the actual image and - of course - a *Neural-NetPad* for the perceptron network at the end of the chain. Another "normal" sprite named *Alberto* will control the operations. To make the model easier to use, we add some buttons as well as a *pen* to make the interface clearer. In the screen shot, the image to be analyzed is at the top of the box, while the neural network displays its result at the bottom. In between, the various intermediate layers are scrolled through and displayed from top to bottom. As a bonus, the model includes the possibility to draw your own numbers. It should be noted that the convolution operations "search vertical line" or "search horizontal line" are not well suited for digit recognition, but they are well comprehensible in the image.

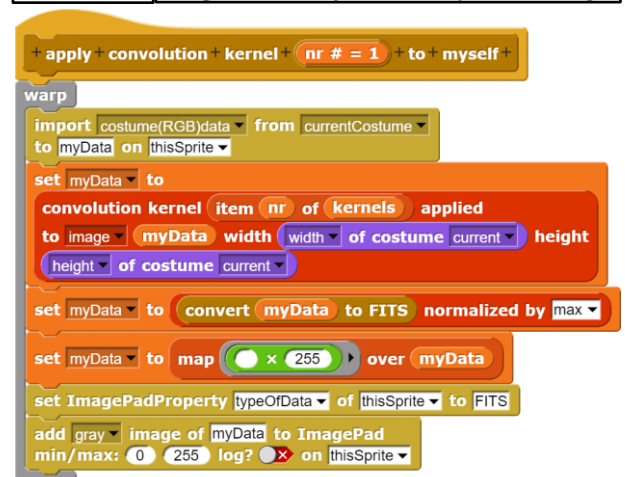
Our CNN is trained with 10 digits each from 64x64 pixels for the zeros and ones. Then it is supposed to "recognize" these as well as other handwritten ones. Actually, we would have to train several kernels of our CNN specifically for this task. Instead, we take only two known kernels for the recognition of vertical and horizontal lines, because by the restriction to two everything can be displayed on the screen and the results are even halfway interpretable. (The recognition rate, however, suffers heavily from this!) Thus, only the perceptron network with four input values is trained.

In the adjacent image, after two stages of reduction, four feature maps of 16x16 pixels each are left, each of which has been run through the operations *Convolution* → *ReLU* → *Max-Pooling* twice: on the far left with the kernel for vertical lines, then with both kernels in different order, and finally twice with the kernel for horizontal lines. The numbers below indicate the mean value of the brightness measured over the entire image. If we apply this to different digits, then the possibility of measuring differences between zeros and ones becomes apparent, despite the very simple procedure.

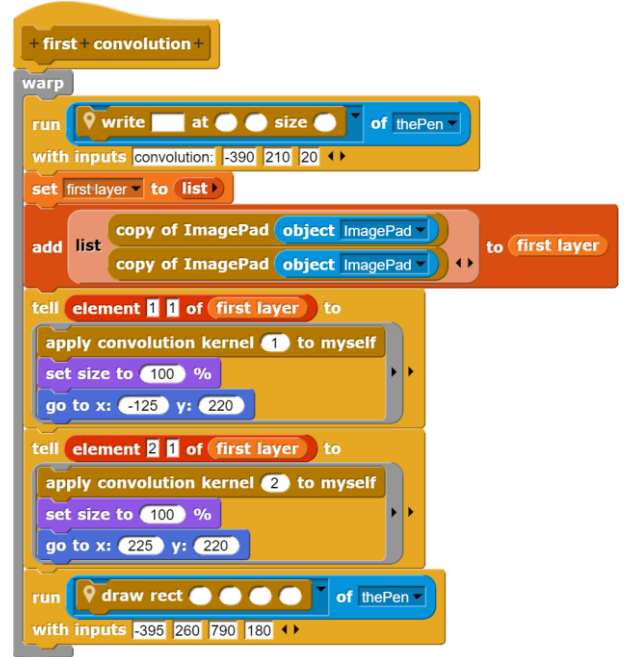
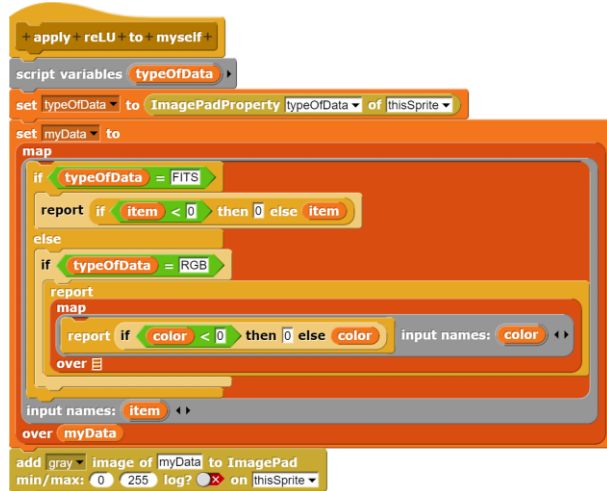


Let's look at the functionalities of the individual objects:

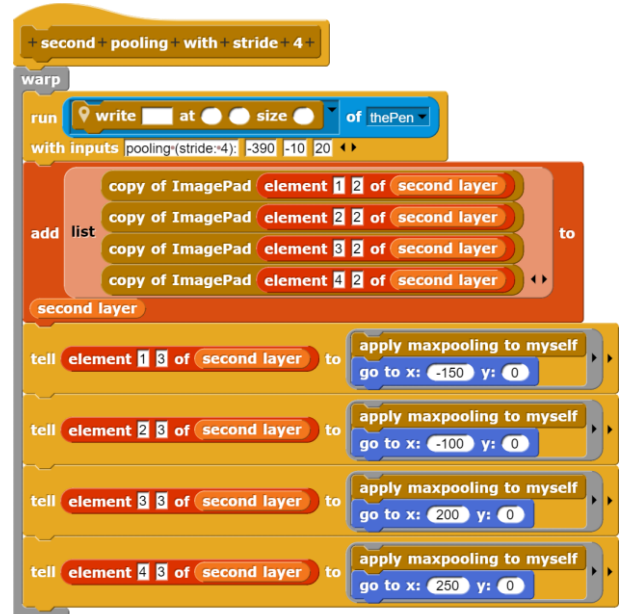
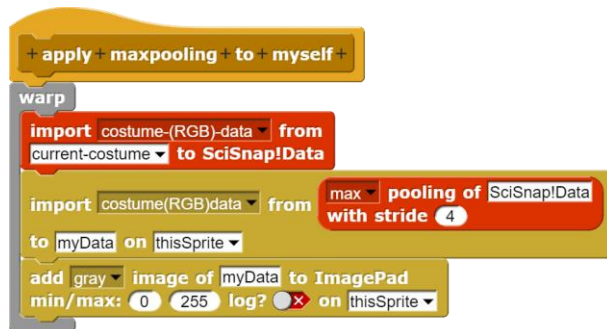
The *ImagePad* provides the data of a new costume as a basis for analysis. To do this, it creates a "first layer" as a list *first layer*, which consists of two copies of itself. On each of these it applies a convolution with two different kernels. After that a few lines are drawn.



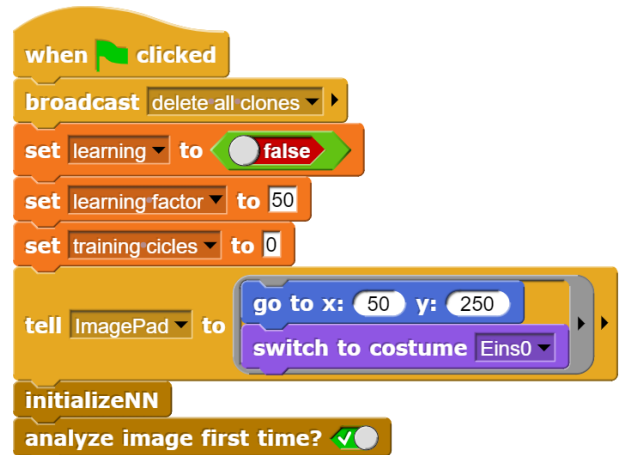
After that, each copy must pass through a *ReLU (rectified linear unit)*, which serves as an activation function. In this case, negative values are simply set to zero.



Finally, a pooling operation is performed to reduce the amount of data. As an example the pooling operation with the four sprites of the second level is given.



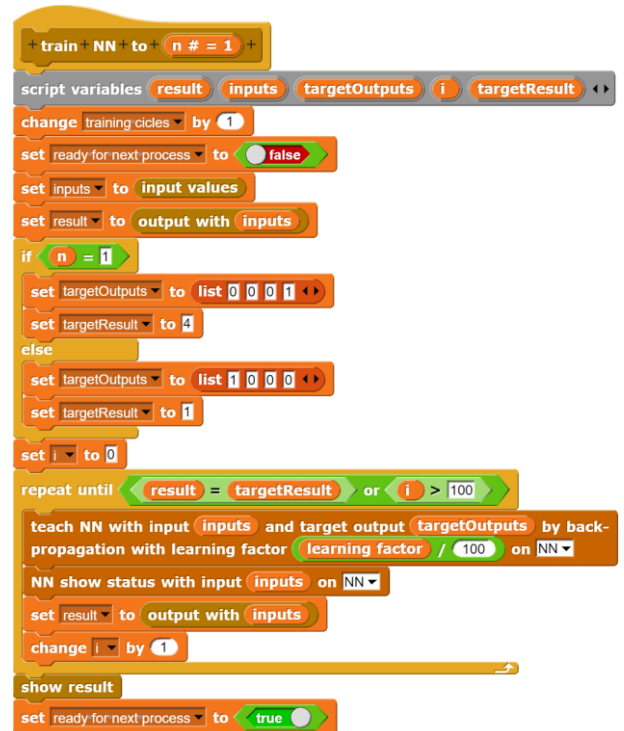
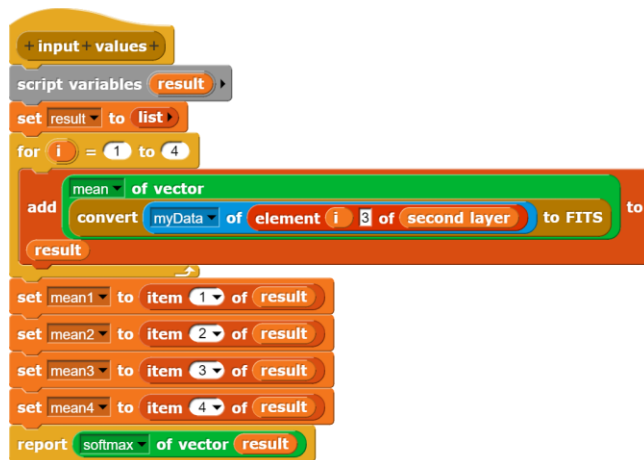
Alberto, as the controller of the whole thing, has to ask the *ImagePad* to change the costume and analyze it afterwards. In doing so, he strictly adheres to the specifications for CNN's.



The *initialize* method only takes care of drawing the lines on the stage. The other methods work with two layers of the CNN, *first layer* and *second layer*, each containing the versions of the digits that appear on the stage. So that they do not interfere with each other, they work with copies of the *ImagePad*, not clones.

After the required copies have been created, they are asked by *Alberto* to perform the respective CNN operation. Finally, the clones of the last level, which are now quite small (4x4 pixels), are displayed in a greatly enlarged form as "final feature maps". These are used to train the neural network.

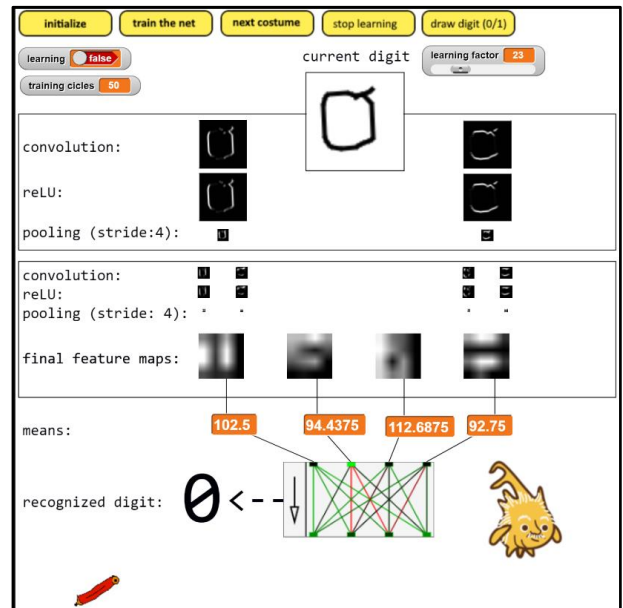
The neural network in the form of a *NeuralNetPad* is supposed to generate the largest output at output 1 for zeros and at output 4 for ones. This is of course completely arbitrary. The current output value is determined by the function *output with <input>*. With its components the net can be trained, if we succeed in determining the average values from the last level of second layer. We still model these suitably with the *softmax* function.



And - did the network learn anything?

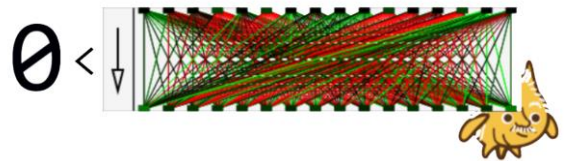
Let's write a number:

Had luck!



Tasks:

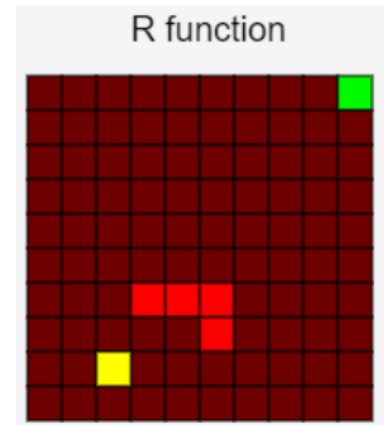
1. Generate a list of 16 values from the *final feature maps*.
2. Analyze this list by a neural network of width 16.
3. Test whether the recognition rate increases, especially of newly written digits. If yes: how do you justify the effect?
4. Also experiment with multilayer neural networks.



11.13 Reinforcement learning / Q learning

In *reinforcement learning*, an *agent* learns from experience, i.e., it performs *actions* that are followed by *reactions*. From these he learns. The reactions must be available for each state of the agent in its environment and each of its actions at the moment when the agent performs the corresponding action. They are usually called *rewards*, and they can be positive or negative. As a whole, they define the goal of the learning process. One way of reinforcing learning is Q-learning. Check out the literature on *reinforcement learning*, *Bellman's optimality principle*, *Markov chains*, and *Q-learning*.

In our example, we want to let a small *robot* (as a *yellow square*) drive around in an environment (realized as a grid) as an agent, where it should drive to a charging station (*green square*) when needed without plowing through the tulip bed (*red squares*). We formulate this goal using a reward function (*R-function*) that holds a reward for each grid element that the robot gets when it enters the field. For "normal" fields we define the reward as -1 (because of the necessary travel distance), for the charging station as $+10$ (sic!) and for the tulip bed as -15 (because there is a lot of trouble when entering). If the robot tries to leave the garden, it gets -10 as a penalty, but stays in the field.

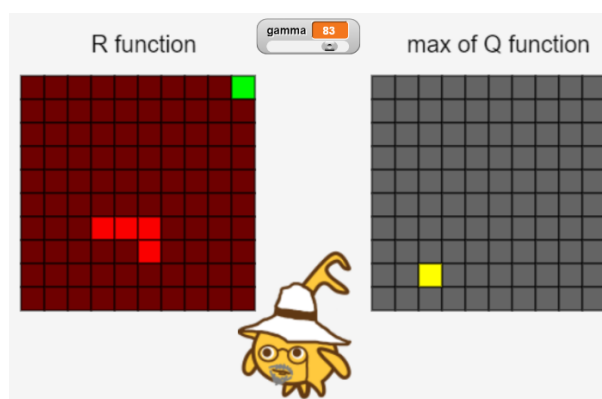


We limit the actions that the robot can perform to steps to the *left*, *right*, *up* and *down* (stored in that order). For each of these actions we now have to decide what happens. We choose as *strategy* the *Q-function*, which tries to maximize the robot's *profit*, i.e. to keep negative rewards small and to collect positive ones. In each case, the robot chooses the action that promises it maximum profit. Please refer to the literature for more information!

$$Q(\text{state}, \text{action}) = \text{reward}(\text{state}, \text{action}) + \gamma \cdot Q(\text{follow-up state}, \text{optimal action in follow-up state})$$

(γ is an damping factor between 0 and 1)

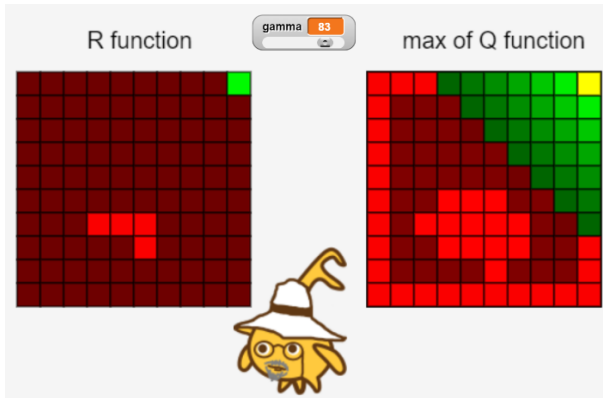
We realize our project in a 10x10 grid, which we display on the stage, configured as ImagePad. Next to the R-function we draw the (positive or negative) maximum value of the four Q-values in the same color representation. We transfer the robot to the side of the Q-function.



```

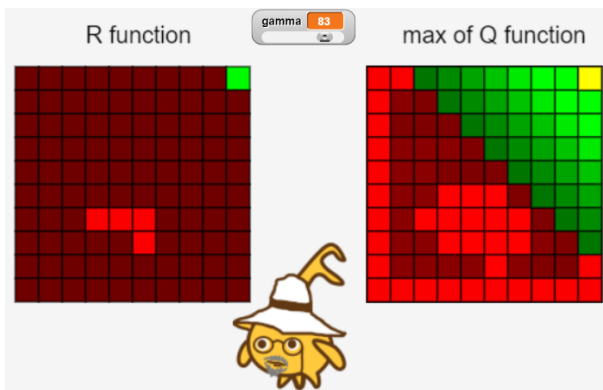
+ Init +
warp
clear
configure theStage as an ImagePad width: 400
height: 300 color: 245 245 245
draw text R-function at 180 150 height: 20
horizontal? on theStage
draw text max-of-Q-function at 450 150 height: 20
horizontal? on theStage
set Rfunction to 20 x 20 table initialized with -1
set element 10 1 of Rfunction to 10
set element 4 7 of Rfunction to -15
set element 5 7 of Rfunction to -15
set element 6 7 of Rfunction to -15
set element 6 8 of Rfunction to -15
show R values starting at 100 150
set Qfunction to 20 x 20 table initialized with list 0 0 0 0
show max Q values starting at 400 150
  
```


Roby now has to explore his environment, i.e. he walks around the grid and calculates the optimal Q-values, as far as this is possible with the previous data. So that this does not take too long, we let him run through the rows of the grid in a loop starting at the top right. After one pass, the Q function looks like this.



You can see that the information about the positive reward at the top-right has "dissipated" from there in the lattice, i.e. has spread weakened to the surrounding cells. In their Q-function is now stored the information in which direction Roby should move in order to succeed. Also the surroundings of the tulip bed were marked as extremely unpleasant, and there, where nothing else is going on, at least the negative experiences with the edge are recorded.

Let's do another pass!



The positive news has continued to spread slowly (because of the damping factor), while the rest of the field cannot get any worse.

We can add a few more of these searches to it.

```

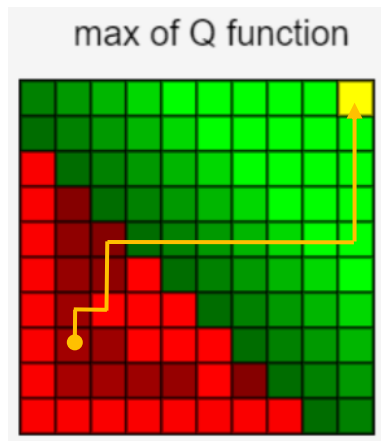
+ roby explores the grid starting at x # = 1 + y # = 1 +
script variables rewards direction
set direction to 1
forever
  show max Q values starting at 400 150
  show roby at x y
  warp
  set rewards to list
  if x = 1
    add 10 to rewards
  else
    gamma / 100 x
    add max of vector element x - 1 y of Qfunction +
      element x - 1 y of Rfunction to
      rewards
  if x = 10
    add 10 to rewards
  else
    gamma / 100 x
    add max of vector element x + 1 y of Qfunction +
      element x + 1 y of Rfunction to
      rewards
  if y = 1
    add 10 to rewards
  else
    gamma / 100 x
    add max of vector element x y - 1 of Qfunction +
      element x y - 1 of Rfunction to
      rewards
  if y = 10
    add 10 to rewards
  else
    gamma / 100 x
    add max of vector element x y + 1 of Qfunction +
      element x y + 1 of Rfunction to
      rewards
  set element x y of Qfunction to rewards
  warp
  if direction = 2
    change x by 1
  else
    change x by -1
  if x > 10
    set x to 10
    set direction to 1
  if y > 9
    roby explores the grid starting at 10 1
  else
    change y by 1
  if x < 1
    set x to 1
    set direction to 2
  if y > 9
    roby explores the grid starting at 10 1
  else
    change y by 1

```

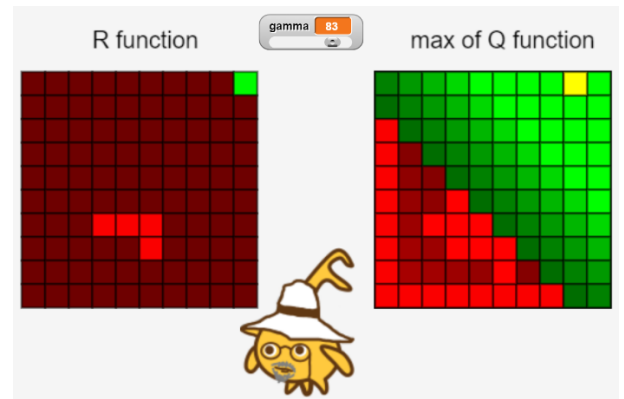

After several passes, nothing changes in this representation. The Q function is constructed. We can now test whether Roby can quickly find its way to the charging station without causing major damage on the way.

The command **roby looks for power starting at 2 8**

results in the following path:



So Roby has learned to get to his destination quickly without causing any damage. Hilberto is happy about the little one!



```

+ roby + looks + for + power + starting + at + x # = 1 + y # = 1 +
script variables rewards direction
repeat until x = 10 and y = 1
  show max Q values starting at 400 150
  show roby at x y
  set direction to maxpos of vector element x y of Qfunction
  warp
  if direction = 1 and x > 1
    change x by -1
  if direction = 2 and x < 10
    change x by 1
  if direction = 3 and y > 1
    change y by -1
  if direction = 4 and y < 10
    change y by 1
  show max Q values starting at 400 150
  show roby at x y

```

Notes

1. *SciSnap!* is not made for small displays, but it runs fine on a larger monitor. 😊
2. The examples in this script are mainly intended to show different ways of using the *SciSnap!* libraries. It is not their task to give examples for good teaching, but hopefully they give hints on which level to work.
3. Accordingly, this script largely lacks examples that the learners can use to find and work on their own problem areas and solutions. If you have any "best-practice" examples, I would be grateful if you could point them out to me. Perhaps a collection of them could be created.
4. The libraries certainly still contain errors and possibilities for improvement. I would also be grateful for hints on this.

For the rest, go for it!

References and sources

- [ABELSON] Abelson, Sussman, Sussman: Structure and Interpretation of Computer Programs, MIT Press
- [Census] <https://archive.ics.uci.edu/ml/datasets/census+income>
- [DBV] Burger, W., Burge, M.-J.: Digitale Bildverarbeitung – Eine Einführung mit Java und ImageJ, Springer 2006
- [FITS] de.wikipedia.org/wiki/Flexible_Image_Transport_System
- [HOU] Hands-On Universe: handsonuniverse.org/
- [HR] <https://studylibde.com/doc/2985884/hertzsprung-russell--und-farb-helligkeits>
- [JSON] Popular Baby Names: <https://catalog.data.gov/dataset/most-popular-baby-names-by-sex-and-mothers-ethnic-group-new-york-city-8c742>
- [NYcitibike] <https://www.citibikenyc.com/system-data>
- [SchulAstro] www.schul-astronomie.de
- [SQL] Modrow, Eckart: Computer Science with Snap!,
<http://ddi-mod.uni-goettingen.de/ComputerScienceWithSnap.pdf>
- [UniGOE] Institut für Astrophysik, Universitaet Goettingen
- [Wolfram] Wolfram, Stephen: A new kind of science, Stephen Wolfram LLC, 2002