Programming with SciSnap!
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. In addition to the book, the full listings of the programs described are available when you send an email to the address below and pay 20 € to the PayPal account listed below. In addition to the book, the full listings of the programs described are available when you send an email to the address below and pay 20 € to the PayPal account listed below. The scripts are developed with Snap! 6.7.3 Build Your Own Blocks.

The libraries and this script can be loaded at

http://emu-online.de/SciSnap.zip or

http://emu-online.de/ProgrammingWithSciSnap.pdf

The SciSnap!-starter can be found at

https://snap.berkeley.edu/snap/snap.html#present:Username=emodrow&ProjectName=SciSnap!-Starter

SciSnap! itself at

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

Prof. Dr. Modrow, Eckart:
Programming with SciSnap!
© emu-online Scheden 2021
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 diagrams \(^3\) (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

\(^1\) 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.
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 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". 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. 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", social networks, friendships or customer-supplier relationships. Teaching has become more relevant.

No one will need all of SciSnap!'s rather extensive libraries at the same time. If you assign even one library to the existing, already quite full palettes, an "overflow" will happen. So it's another goal of SciSnap! to keep order and overview in the command palettes of Snap!. For this purpose is the ability of SciSnap! to create, delete or hide palettes (in Snap!: categories). The required libraries are sorted into these palettes.

2. Therefore, another goal of SciSnap! is to expand the Snap!-palettes, and if necessary, to reduce them, thus creating Snap! versions configured for different purposes.

---

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
I you change Snap!, then you disconnect yourself from the current development of this fantastic tool. I have therefore refrained from making direct changes to the Snap! code. Instead, I have divided SciSnap! into two parts: in the first, the SciSnap!-Starter, a configuration is created. The commands to generate it are used as a normal Snap! project. From this configuration the actual SciSnap! is started. Use the Snap! cloud, and both are fast.

3. A goal of SciSnap! is to work with the current Snap! version - as long as there are no fundamental changes.

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.

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

Snap! is not only a fantastic development tool, but it is based on a fantastic concept. As a graphical re-implementation of MIT’s Scheme language, based on the "CS Bible" Structure and Interpretation of Computer Programs 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...

5. 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" and the Snap!-fork SQL-Snap!. They were supplemented by numerous mathematical operators, SQL, sketchpads, neural networks and graphs. A detailed description of Snap! with a lot of examples can be found at "Computer Science

---

7 You can include starter projects in the favorites bar and run them with one click, e.g. as https://snap.berkeley.edu/snap/snap.html#present:Username=emodrow&Project-Name=SciSnap!-Starter&editMode
8 https://en.wikipedia.org/wiki/MIT/GNU_Scheme
10 as well as other materials on http://emu-online.de
11 http://snapextensions.uni-goettingen.de/
with Snap!" and of course, in the Snap!-manual. 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 will take care of the more scientific applications, Hilberto of mathematical and data-oriented ones. If one is active, the other can rest a bit. Both claim 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.

I hope you enjoy working with SciSnap!.

Göttingen, on 2021.7.1

12 http://ddi-mod.uni-goettingen.de/InformatikMitSnap.pdf
14 he works in astrophysics with Rick Hessman
15 https://de.wikipedia.org/wiki/David_Hilbert
Content

Preface .................................................................................................................................. 3

Content .................................................................................................................................. 7

1 SciSnap!-Starter .................................................................................................................. 9
   1.1 Create start configurations ......................................................................................... 9
   1.2 New blocks in the standard palettes of Snap! ......................................................... 11

2 The structure of the SciSnap!-Sprites ............................................................................... 13

3 The SciSnap!-Libraries ...................................................................................................... 15
   3.1 The Mathematics-Libraries (SciSnap!FullMathLibrary.xml) .................................. 15
       3.1.1 Complex Numbers (SciSnap!ComplexNumbersLibrary.xml) .................... 15
       3.1.2 The MathPad (SciSnap!MathPadLibrary.xml) ............................................ 16
       3.1.2 Linear Algebra (SciSnap!LinearAlgebraLibrary.xml) ................................ 17
       3.1.3 Statistics (SciSnap!StatisticsLibrary.xml) .................................................... 19
       3.1.4 Sets (SciSnap!PredicateSetsLibrary.xml) ....................................................... 20
       3.1.5 Numerical Methods (SciSnap!NumericMathLibrary.xml) ............................ 22
   3.2 The Data-Library (SciSnap!DataLibrary.xml) .......................................................... 23
   3.3 The SQL-Library (SciSnap!SQL-Library.xml) .......................................................... 28
   3.4 The ImagePad-Library (SciSnap!ImagePadLibrary.xml) ......................................... 31
   3.5 The PlotPad-Library (SciSnap!PlotPadLibrary.xml) ............................................. 33
   3.6 The GraphPad-Library (SciSnap!GraphPadLibrary.xml) ....................................... 35
   3.7 The NeuralNetPad-Library (SciSnap!NNPadLibrary.xml) ..................................... 37

4 Data Import and Export ................................................................................................... 38

5 Examples and Tasks ......................................................................................................... 42
   5.1 Representation of complex numbers ..................................................................... 42
   5.2 Affine transformation of a triangle in $\mathbb{R}^2$ ................................................. 43
   5.3 Rotation of a pyramid in $\mathbb{R}^3$ ........................................................................ 44
   5.4 Graph of normal distribution ............................................................................. 45
   5.5 Cartesian product of three sets ........................................................................... 46
   5.6 Representation of a set of points and the regression line ..................................... 47
   5.7 Interpolation polynomial through n points ....................................................... 48
   5.8 Approximation of a tangent by secants ............................................................. 50
   5.9 Finite series .......................................................................................................... 52
   5.10 Application of the Taylor series to the mathematical pendulum ..................... 54
   5.11 Fourier expansion for a square wave signal with numerical integration .......... 57
   5.12 NY Citibike Tripdata 1: Correlations .................................................................. 61
   5.13 Income data from the US Census Income Dataset ............................................ 62
   5.14 NY Citibike Tripdatea 2: data processing ......................................................... 64
   5.15 Under- and Overfitting ....................................................................................... 65
   5.16 NY Citibike Tripdata 3: World Map Library .................................................... 68
   5.17 Star spectra .......................................................................................................... 72
   5.18 Classification in the HR diagram according to the kNN method ...................... 75
   5.19 Data import and export: CSV import ............................................................... 38
5.20 Data import and export: JSON import ........................................ 39
5.21 Data import and export: writing CSV and text data to a file .......... 41
5.22 Drawing a function and its derivatives .................................... 77
5.23 Data plot of random data scattering around a function graph ........ 78
5.24 Histogram of random values .................................................. 79
5.25 Covid-19 data analysis ......................................................... 80
5.26 Shadow lengths in the lunar crater Tycho ................................ 82
5.27 Plot of mixed data ............................................................. 83
5.28 Simple SQL query .............................................................. 84
5.29 More complex SQL query .................................................... 84
5.30 Data import and export: SQL import ....................................... 39
5.31 Dealing with the SQL library .................................................. 85
5.32 Random graphic ............................................................... 86
5.33 False color image of a lunar crater ......................................... 87
5.34 Slice through the image of the lunar crater Tycho ...................... 87
5.35 Data import and export: false color image of Saturn .................. 38
5.36 Data import and export: data import with the mouse ................. 40
5.37 Plot of image data as histogram ............................................. 88
5.38 Simulation of a planetary transit in front of a sun ....................... 89
5.39 Affine transformation of an image ......................................... 90
5.40 Kernel applications for edge detection in images ....................... 91
5.41 Mean distances in a random graph (Small Worlds) ................. 92
5.42 Mean distances in a scalefree graph (Small Worlds) ............... 92
5.43 Histogram "edges per vertex" in a random graph ..................... 93
5.44 Histogram "edges per vertex" in a scalefree graph ................... 93
5.45 Breadth and depth search in the family tree ........................... 94
5.46 A simple perceptron as a graph .......................................... 96
5.47 A simple learning perceptron as a graph .............................. 98
5.48 Training of a neural network ............................................... 100
5.49 Traffic sign recognition with a neural network ....................... 101
5.50 Character recognition with a Convolutional Neural Network ........ 106
5.51 k-means-Clustering ......................................................... 112
5.52 DNA-Relatedness und Levenshtein-Distance ............................ 115

Notes ......................................................................................... 116

List of examples ........................................................................... 117

References and sources ................................................................ 119
1  SciSnap!-Starter

1.1  Create start configurations\textsuperscript{16}

As long as \textit{Snap!} itself does not yet work with changeable palette configurations, you have

to deal with self-made solutions - if you want to have something like that\textsuperscript{17}. Since I also

want to work with the current \textit{Snap!} version, more serious interventions in the actual code

are not allowed. \textit{Snap!} looks at which palette the new blocks are to be sorted into when

loading a project. If this palette is missing, the blocks are ignored. We therefore have to

work in two steps to create \textit{SciSnap!} applications:

1. New palettes and some global variables \textit{SciSnap!} works with

   are created. Not needed palettes can be hidden. If you want,

   you can load the needed libraries into the corresponding palettes right after. You should add a "green flag block" to which

   the start commands will be attached. This starter configuration is saved as a \textit{Snap!} project, e.g. in the cloud.

2. You should include a link to "your" \textit{SciSnap!} starter project in the

   favorites bar in a form that executes the "green flag commands" immediately: e.g. as

   \url{http://snap.berkeley.edu/snap/snap.html

   #present:Username=emodrow&ProjectName=SciSnap!-starter&editMode}

   When you add the adjacent JavaScript block to your script, the

   normal "Open Project" dialog immediately follows.

3. Work is done in this configuration and the current project is

   again stored in the cloud.

4. If you want to continue working on the current project later, then load the Starter pro-

   ject and with it the current \textit{Snap!} version, e.g. with the link in the favorites bar. After

   its execution, the process continues as usual. If both files are in the cloud, then the

   process is simple and fast.

5. If you want to start a new project with "New" from the File menu, the palettes are kept,

   but the \textit{SciSnap!} blocks have to be reloaded.

\textsuperscript{16} Due to the current problems with JavaScript, JavaScript must first be enabled in the tools

menu.

\textsuperscript{17} To be clear, using the import library block, you can also load the \textit{SciSnap!} libraries into any

other palette. So you don't need to create new palettes!
We will go through this with an example: The goal is to work with the SQL blocks and additionally insert a palette for your own blocks. The palettes for sound and the pen are not to be used.

Step 1: We load Snap! and then the general SciSnap! starter file. (We could also import the SciSnap! GlobalBlocks library instead). We get the adjacent screen. There we enable JavaScript if necessary.

Step 2: We specify which palettes are to be hidden (here: Sound and Pen) and which are to be created with which names and colors (here: Sql and MyBlocks). In this case, the SQL library should also be loaded into the “Sql” palette at once. This project is first saved under the name “SciSnap!-SQLStarter”.

Step 3: We save the following link as a favorite: (without spaces and line feeds):

```
http://snap.berkeley.edu/snap/snap.html
#present:Username=emodrow&Project-Name=SciSnap!-SQLStarter&editMode
```

After that it will be executed.

As a result we get a SQL working environment.
## 1.2 New blocks in the standard palettes of **Snap!**

The additional blocks of the *Looks* palette are there because they have to be assigned to a standard *Snap!* palette to be loaded. The *Looks* palette is where they fit best and are the least disruptive. There are the following blocks:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch to SciSnap!</td>
<td>Creates the <em>SciSnap!</em> logo and displays it, generates the desired palettes. Enlarges the stage to 800x600 pixels.</td>
</tr>
<tr>
<td><strong>SciSnap! global property</strong> minValue &lt;br&gt; set SciSnap! global property minValue to</td>
<td>Reads a property.</td>
</tr>
<tr>
<td>set SciSnap! global variables and properties</td>
<td>Allows you to modify or create a property.</td>
</tr>
<tr>
<td>add category newCategoryName color</td>
<td>Inserts a palette with the named name and RGB colors mentioned.</td>
</tr>
<tr>
<td>hide category categoryName</td>
<td>Hides, shows, or deletes a palette. Snap's standard palettes will not be deleted.</td>
</tr>
<tr>
<td>import library to category Looks</td>
<td>Imports a library of any name into the specified palette. The library is selected with the mouse.</td>
</tr>
<tr>
<td>change SpriteName to newName</td>
<td>Changes the name of the current sprite.</td>
</tr>
<tr>
<td>show global message title: headline message: theMessage</td>
<td>Creates a message window in the center of the screen.</td>
</tr>
<tr>
<td>report error something-wrong</td>
<td>Outputs an error, if possible, via the current sprite and enters it into <em>SciSnap!Messages</em>.</td>
</tr>
<tr>
<td>copy of costume</td>
<td>Provides a copy of the current costume.</td>
</tr>
<tr>
<td>copy of costume <em>my costume</em></td>
<td>Provides the costume of a sprite, e.g. for pooling operations.</td>
</tr>
<tr>
<td>costume of</td>
<td></td>
</tr>
</tbody>
</table>

In addition, the following blocks for handling sprites can be found in the *Control* palette:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create a duplicate of with name spriteName</td>
<td>Creates a duplicate of the specified sprite with the specified name.</td>
</tr>
<tr>
<td>create a permanent clone of with name spriteName</td>
<td>Creates a permanent clone of the specified sprite with the specified name.</td>
</tr>
<tr>
<td>Import Sprite</td>
<td>Imports a locally stored sprite.</td>
</tr>
</tbody>
</table>
The **Operators** palette contains additional mathematical operators and string functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>random</code></td>
<td>Returns a random number between 0 and 1.</td>
</tr>
<tr>
<td><code>n</code></td>
<td>Returns π.</td>
</tr>
<tr>
<td><code>e</code></td>
<td>Returns the Euler number e.</td>
</tr>
<tr>
<td><code>round (1.2357 to 2) digits</code></td>
<td>Returns the specified number rounded to the specified number of decimal places.</td>
</tr>
<tr>
<td><code>5!</code></td>
<td>Returns n!</td>
</tr>
<tr>
<td><code>(5)</code></td>
<td>Returns the binomial coefficient.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>is a vector</code></td>
<td>Some type checks.</td>
</tr>
<tr>
<td><code>substring of thisString from 1 to 4</code></td>
<td>Returns a partial string of strings.</td>
</tr>
<tr>
<td><code>delete all this in thisString all first</code></td>
<td>Deletes all or the first string(s) in a string.</td>
</tr>
<tr>
<td><code>upper case thisString</code></td>
<td>Returns a string in capital letters.</td>
</tr>
<tr>
<td><code>lower case thisString</code></td>
<td>Returns a string in lowercase.</td>
</tr>
<tr>
<td><code>write text this text to TXT-file this file</code></td>
<td>Writes a string to a text file in the browser's download folder.</td>
</tr>
<tr>
<td><code>index of thing in thisString</code></td>
<td>Returns the position of a substring.</td>
</tr>
<tr>
<td><code>replace all these with that in thisString all first</code></td>
<td>Replaces all or the first string(s) in a string.</td>
</tr>
<tr>
<td><code>get label from text-data SciSnapData at column 1 max. textwidth 5 column spacing 2</code></td>
<td>Reads a column heading from a table, e.g. to chart label.</td>
</tr>
<tr>
<td><code>datetime: 2021-07-02T10:08:14</code></td>
<td>Conversions from <strong>datetime</strong> to the specified values.</td>
</tr>
</tbody>
</table>

In addition, the following variables can be found in the variable palette and in the sensing palette a block for standardised time:

Other variables are created depending on the use of the libraries when configuring the sprites.
2 The structure of the SciSnap!-Sprites

The second part of SciSnap! consists of independent libraries, which on the one hand can be used generally in Snap! scripts (Math, Data, SQL), and on the other hand work with specially configured sprites (MathPad, NeuralNetPad, GraphPad, PlotPad, ImagePad). The reason for this are the properties, which are very different for a NeuronalNetPad than for a PlotPad. In addition, such "special sprites" must have their own data areas in which e.g. image data can be stored. A global data area can be found in the SciSnap!Data, with which the blocks of the Data library work 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 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 query 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 linear systems of 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: several Math libraries for different areas of mathematics (60 blocks), a data library (30 blocks) for handling the actual data, an ImagePad for image processing (19 blocks), a PlotPad for graphical representations (24 blocks), a NeuralNetPad for perceptron networks, an SQL library for database queries (27 blocks) and a GraphPad for graph theory applications. 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:

Using the libraries, they import data from
- Image-files
- Text-files
- SQL-queries
- JSON-files
- CSV-files
- ...

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.
3 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. For loading, it is a prerequisite that these palettes are available. If a library is to be loaded into another palette, then the block `import library to category` can be used for this. With its help, the libraries can also be used in a "normal", i.e. unmodified Snap! version.

3.1 The Mathematics-Libraries
(SciSnap!FullMathLibrary.xml)

3.1.1 Complex Numbers (SciSnap!ComplexNumbersLibrary.xml)

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:

<table>
<thead>
<tr>
<th>complex 2 + 3 * i</th>
<th>Returns a complex number in Cartesian form.</th>
</tr>
</thead>
<tbody>
<tr>
<td>complex 2 * e^i 30</td>
<td>Returns a complex number in polar form.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>complex Cartesian style</th>
<th>Returns a complex number in Cartesian form.</th>
</tr>
</thead>
<tbody>
<tr>
<td>complex polar style</td>
<td>Returns a complex number in polar form.</td>
</tr>
<tr>
<td>( \text{complex } \text{real-part } ) of ( \text{complex } \text{complex } 2 + 3 * i ) polar style</td>
<td>Example of format conversion.</td>
</tr>
<tr>
<td>( \text{complex } \text{real-part } ) of ( \text{complex } \text{complex } 2 + 3 * i ) polar style</td>
<td>The components of a complex number can be accessed - regardless of their format.</td>
</tr>
<tr>
<td>( \text{complex } ) + / -</td>
<td>And, of course, you can calculate with them.</td>
</tr>
</tbody>
</table>

Example of multiplication: \( \text{complex } \text{complex } 2 + 3 * i \) and \( \text{complex } 2 \cdot e^i 30 \).
3.1.2 The MathPad (SciSnap!MathPadLibrary.xml)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configure sprite [thisSprite] as a MathPad</td>
<td>Configures a sprite as a MathPad and draws a 3-dimensional coordinate system centered in the middle.</td>
</tr>
<tr>
<td>is thisSprite a MathPad?</td>
<td>Test for MathPad configuration.</td>
</tr>
<tr>
<td>set MathPadProperty costumeProperties of thisSprite to ␣</td>
<td>Sets a MathPad property.</td>
</tr>
<tr>
<td>MathPadProperty costumeProperties of thisSprite</td>
<td>Reads a MathPad property.</td>
</tr>
<tr>
<td>set MathPad costume properties width: 400</td>
<td>Sets the costume properties of a MathPad to the specified values.</td>
</tr>
<tr>
<td>height: 300 color: 245 245 245</td>
<td></td>
</tr>
<tr>
<td>offsets: 0 0 on thisSprite</td>
<td></td>
</tr>
<tr>
<td>set MathPad properties lineWidth: 1 onlyPoints? 10 maxValue: 0 startPoint: 0 0 0 on thisSprite</td>
<td>Sets the line properties of a MathPad to the specified values. If &quot;onlyPoints&quot; is set, only the endpoints are drawn.</td>
</tr>
<tr>
<td>draw centered axes to a MathPad on thisSprite</td>
<td>Draws the coordinate system on a MathPad.</td>
</tr>
<tr>
<td>draw vector [vector] color: 255 255 255 on MathPad complex-number</td>
<td>Draws a vector, complex number, line or object described by a list of vectors.</td>
</tr>
<tr>
<td>line-to object-to</td>
<td></td>
</tr>
<tr>
<td>affine transformation of ( \mathbf{v} ) by ( \mathbf{A} \rightarrow \mathbf{B} ) for MathPad</td>
<td>Performs an affine transformation in the plane for a list of points, e.g. an object, described by mapping three points to three others.</td>
</tr>
</tbody>
</table>
3.1.3 Linear Algebra (SciSnap!LinearAlgebraLibrary.xml)

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

- **vector**
  - Returns a vector of any dimension with predefined values.

- **vector dim**
  - Returns a vector of arbitrary dimension with random values from the specified range.

- **matrix of vectors**
  - Returns the matrix from the specified vectors.

- **3 x 2 matrix with random components range 1 to 10**
  - Returns a matrix of arbitrary dimension with random values from the specified range.

- **transpose**
  - Matrices and vectors can be transposed.

- **linear operation**
  - Between scalars, vectors and matrices the respective allowed operations can be performed. Since vectors can be processed with the standard arithmetic operators of Snap! there are no separate blocks for them.

- **polynomial interpolation for points**
  - Calculates the coefficients of the polynomial through \( n \) points.

- **apply \( \square \) to points \( \square \)**
  - Applies a matrix to a list of points (look at the example).
Calculates the solution to a system of linear equations.

The block returns several data about the passed matrix: the diagonalized matrix, if any, its rank, whether column swaps have occurred, and the current order of the columns.

If only the diagonalized matrix is of interest, then we consider the first element of the result.

Beispiele:
### 3.1.3 Statistics (SciSnap!StatisticsLibrary.xml)

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.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binomial distribution</td>
<td>( b(N, p, k) = \binom{N}{k} \cdot p^k \cdot (1-p)^{N-k} )</td>
</tr>
<tr>
<td>Binomial probability</td>
<td>( h(N, M, n, k) = \binom{M}{k} \cdot \binom{N-M}{n-k} / \binom{N}{n} )</td>
</tr>
<tr>
<td>Poisson distribution</td>
<td>( p(\theta, k) = \frac{\theta^k \cdot e^{-\theta}}{k!} )</td>
</tr>
<tr>
<td>Pareto distribution</td>
<td>( \text{pareto}(x_{\text{min}}, k, x) = \frac{k \cdot x_{\text{min}}^k}{x^{k+1}} ) if ( x \geq x_{\text{min}} ); 0 sonst</td>
</tr>
<tr>
<td>Normal distribution</td>
<td>( n(x, \mu, \sigma) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} / \sqrt{2\pi\sigma^2} )</td>
</tr>
</tbody>
</table>
3.1.4 Sets (SciSnap!PredicateSetsLibrary.xml)

Sets are implemented in SciSnap! as three-element lists. There are two versions of this. In the first one, predicates (x<5) are used to enter ranges next to an enumeration of elements. In the second, this is done using intervals (3<x<7). The libraries differ only in the definition of a set and an additional block for interval compression. The first element of a set contains the type ("set"), the second either a predicate or it is empty or a list of intervals. The third element contains a list of elements not yet covered by the predicate or intervals. Compound predicates are again lists, which contain the Boolean operator ("NOT", "OR", "AND") in the first element and afterwards one or two predicates, which again can be compound. Sets of this type can be generated either by enumeration or by specifying a predicate.

For further work with sets, the known set operations are available. If predicates are used, then only numbers and strings are useful as elements. The following operations do not refer to bounded sets in this case. Predicates or interval memberships can be identified with "evaluate with".

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>evaluate [predicate]</code></td>
<td>Returns the result of the predicate applied to the passed element or an interval list.</td>
</tr>
<tr>
<td><code>element ∈ set</code></td>
<td>Returns &quot;true&quot; if the element is an element of the set, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td><code>set1 ∩ set2</code></td>
<td>Returns the intersection of two sets.</td>
</tr>
<tr>
<td><code>set1 ∪ set2</code></td>
<td>Returns the union of two sets.</td>
</tr>
<tr>
<td><code>element ∈ set </code>→ <code>text</code></td>
<td>Represents the elements of a set &quot;a bit more readable&quot;.</td>
</tr>
<tr>
<td><code>text [list, set] </code>→ <code>elements</code></td>
<td>Generates the corresponding list of elements from a text. Lists are bracketed &quot;square&quot; in the text, sets are bracketed &quot;curly&quot;.</td>
</tr>
</tbody>
</table>

The following operations must be performed with finite sets because all elements are processed. For this reason there is an upper bound for set elements, which is set either in the SciSnap!Properties or by the block in the set library.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is set1 ⊆ set2 ?</td>
<td>Returns &quot;true&quot; if the first set is subset of the second, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>is set1 = set2 ?</td>
<td>Returns &quot;true&quot; if the sets contain the same elements, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>set1 × set2</td>
<td>Returns the Cartesian product of two sets.</td>
</tr>
<tr>
<td>10 elements of set</td>
<td>Returns the first n elements of a set as a list.</td>
</tr>
<tr>
<td>merge list of set intervals</td>
<td>Merges a list of intervals.</td>
</tr>
</tbody>
</table>

Because on the one hand the predicates of the sets are so important, and on the other hand the procedures are also needed in Boolean algebra, some more blocks were added to the set library.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 → 0 ?</td>
<td>Implication, returns &quot;true&quot; if the conclusion is formally true, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>1 ↔ 0 ?</td>
<td>Equivalence, returns &quot;true&quot; if the inputs are formally equivalent, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>number ▷ boolean</td>
<td>Conversion of switch values into truth values.</td>
</tr>
<tr>
<td>boolean ▷ number</td>
<td>Conversion of truth values into switch values.</td>
</tr>
</tbody>
</table>

Examples:

- set set1 to set of `{1 5 27 30 44}`
- set set2 to set of `{x | mod 2 = 0}`
- set set3 to set of `{x | -Infinity ≤ x ≤ 3}`

- 10 elements of set1 ∩ set2
- 5 elements of set1 \ set2
### 3.1.5 Numerical Methods (SciSnap!NumericMathLibrary.xml)

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

<table>
<thead>
<tr>
<th>Block Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Calculation of a sequence element.</strong> The term must be entered with (gray) ring (&quot;ringified&quot;).</td>
<td>Example: the 17th element of the sequence $\frac{1}{\sqrt{n}}$.</td>
</tr>
<tr>
<td><strong>Returns the first n members of a sequence as a list.</strong></td>
<td>Example: The first 10 elements of the sequence $\frac{1}{\sqrt{n}}$.</td>
</tr>
<tr>
<td><strong>Calculates the sum of a finite series.</strong> The term must be entered with (gray) ring (&quot;ringified&quot;).</td>
<td>Example: The sum of the first 1000 elements of the series $\sum_{i=1}^{1000} \frac{1}{i^2}$.</td>
</tr>
<tr>
<td><strong>Sequence of secant slopes in a point.</strong> The sequence can also be specified explicitly in the form of a list. The term must be entered with (gray) ring (&quot;ringified&quot;).</td>
<td>Example: 10 secant slopes near the point with $x=2$ of the function $f(x) = x^3 - 3x$, calculated with the sequence $\frac{1}{n^2}$.</td>
</tr>
<tr>
<td><strong>Numerical calculation of an integral using the trapezoidal method.</strong> The term must be entered with (gray) ring (&quot;ringified&quot;).</td>
<td>Example: Calculation of the integral $F = \int_0^\pi \cos(2\pi x) , dx$</td>
</tr>
<tr>
<td><strong>Root calculation according to Newton's method.</strong> The term must be entered with (gray) ring (&quot;ringified&quot;).</td>
<td>Example: Calculation of the root of $f(x) = x^3 - 3x$, start at $x=1$.</td>
</tr>
<tr>
<td><strong>Numerical calculation of the derivative in a point.</strong> The term must be entered with (gray) ring (&quot;ringified&quot;).</td>
<td>Example: Calculation of the derivative of $f(x) = x^3 - 3x$ at $x=0$.</td>
</tr>
</tbody>
</table>
3.2  The Data-Library (SciSnap!DataLibrary.xml)

The data library of SciSnap! serves on the one hand for the direct manipulation also of larger data sets, on the other hand for the evaluation of data, e.g. for the computation of statistical values such as the variance or correlation. Because number structures like vectors and matrices are implemented in the math libraries, the data library is largely limited to tables as an additional structure. Their rows and columns can be identified either by their numbers or by 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.

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

| Returns an empty table (as a start structure for further table operations). |
| Returns a table of the specified size without contents. |
| Returns a table of the specified size without contents, but with column headers. |
| Returns a copy of a list or table. Since Snap! assigns lists as references, you can use this block to avoid unintended changes to the original. |
| The block imports tables, image data or SQL data into the global variable SciSnap!Data, which the other blocks of the library work with by default. Example: Import a table using the file selection dialog. |
| Starts the file selection dialog. |
| Writes a table to a file in the browser’s download area with the specified name. |
transpose table or list

Returns a transposed table or list as a result, i.e. one in which rows and columns have been swapped.

The following blocks are used for direct manipulation of tables.

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>row numberOrName</code></td>
<td>Returns a row or column of the specified table.</td>
</tr>
<tr>
<td><code>columns numberOrName</code></td>
<td>Returns a row or column of the specified table.</td>
</tr>
<tr>
<td><code>from row numberOrName to last</code></td>
<td>Returns the specified row range of the specified columns. <strong>Example:</strong> Results of the three specified parties from 2010 and 2020.</td>
</tr>
<tr>
<td><code>add row column column-headers</code></td>
<td>Adds a row, a column or column headers to the specified table. Missing elements are added with empty content, &quot;overhanging&quot; are ignored.</td>
</tr>
<tr>
<td><code>delete row numberOrName</code></td>
<td>Deletes a row or column from the specified table. <strong>Example:</strong> Deletes the column for the year 2010 from the sample table.</td>
</tr>
<tr>
<td><code>delete column B</code></td>
<td>Deletes a row or column from the specified table. <strong>Example:</strong> Deletes the column for the year 2010 from the sample table.</td>
</tr>
<tr>
<td><code>element numberOrName numberOrName</code></td>
<td>Returns the specified table element. <strong>Example:</strong> Result of the party KKA in 2011.</td>
</tr>
<tr>
<td><code>element #2011 KKA</code></td>
<td>Sets the value in a table at the specified position.</td>
</tr>
</tbody>
</table>

**Example:** The first column of the example table without the heading.
The following blocks are much more powerful. The examples refer again to the example table.

- **Select rows of** `SciSnapData` **where**
  - column `numberOrName` is `less-than`
  - column `numberOrName` is `greater-than`
  - column `numberOrName` is `equal-to`
  - column `numberOrName` is `different-from`

  Returns selected rows of a table that satisfy the specified criterion.

  **Example:** All election results with parties beginning with "A".

- **Select rows of** `SciSnapData` where column `numberOrName` is `less-than`  

  Returns the selected property of a vector, i.e. a row or column consisting only of numbers.

- **Min** of vector
  - min
  - max
  - minpos
  - maxpos
  - number
  - sum
  - mean
  - median
  - variance
  - standard-deviation
  - softmax

  Returns the minimum, maximum, count, sum or average of the first column of data specified, grouped by the second. The headings can be included - or not.

  **Examples:**
  - The mean values for 2010, grouped by party.
  - The mean values of the party AAB, grouped by years.

- **Mean** of column `2010` of `SciSnapData` grouped by column `numberOrName`

  Returns a table without duplicate rows or a vector without duplicates.

  **Example:** We insert the first row of the table again at the end ...

  ...and then delete the duplicates again.

- **SciSnapData** without duplicates

  Returns a table sorted by the specified column in ascending or descending order.
Example: The election results sorted by the year 2011.

Returns ranges, covariance and correlation for two table columns.

Example: Correlation for the years 2010 and 2011.

Normalization of a vector by dividing it by the mean, maximum, number, sum, median of its values or by the softmax function.

Example: Results of the party KKA, "normalized" by the mean.

Returns a table compressed by the factor n by averaging. Texts cannot be compressed in this way, of course.

Returns n points scattering around a straight line, given by slope and y-axis intercept, in a range bounded by "range". Serves mostly for test purposes.

Example:
5 points scattering around \( f(x) = x - 2 \).

Returns n points that scatter around an arbitrary function, given by its "ringified" operators, in a range bounded by "range". Mostly used for testing purposes.

Example:
5 points scattering around \( f(x) = x^2 - 4 \).

Returns the slope and y-intercept of the regression line through the specified data.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression line</td>
<td>Example: Regression line through 10 random points scattering around a straight line.</td>
</tr>
<tr>
<td>max pooling of SciSnap!Data with max or mean</td>
<td>Returns a table that has been compressed by max- or mean-pooling with step size n. The dimensions of the new table are passed before the result. Well applicable to image data. Example: A matrix of 100x100 random numbers is compressed with step size 25.</td>
</tr>
<tr>
<td>5 next neighbors of SciSnap!Data</td>
<td>k-nearest neighbor (kNN) method in two dimensions for machine learning. Example: HR graph</td>
</tr>
<tr>
<td>convolution kernel applied to table SciSnap!Data</td>
<td>Returns the result of a convolution applied either to a table or to image data. Example: Edge detection, CNN</td>
</tr>
<tr>
<td>subsection of RGB-data from numberOrName number to numberOrName number</td>
<td>Returns a section of a table, matrix, list or image data given by the two points &quot;left-up&quot; and &quot;right-down&quot;.</td>
</tr>
<tr>
<td>column 1 of SciSnap!Data name</td>
<td>Returns the row or column number of a table with the specified label or vice versa.</td>
</tr>
<tr>
<td>column number of SciSnap!Data name</td>
<td>Returns the row or column number of a table with the specified label or vice versa.</td>
</tr>
<tr>
<td>10 random points with ranges for x: 0 - 100, y: -100 - 100</td>
<td>Returns n random points from the specified range.</td>
</tr>
<tr>
<td>3 -means clustering for SciSnap!Data with Euclidean metrics</td>
<td>Clustering of n-dimensional data using the k-means method. The Euclidean distance is taken as the metric. cluster numbers are appended to the data.</td>
</tr>
<tr>
<td>3 -means clustering for SciSnap!Data with metric weighted-operator</td>
<td>Clustering with any metric.</td>
</tr>
<tr>
<td>Levenshtein-distance of Saturday and Sunday</td>
<td>Returns the Levenshtein-distance.</td>
</tr>
</tbody>
</table>
### 3.3 Die SQL-Library (SciSnap!SQLLibrary.xml)

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 two global variables `SQLData` and `SQLProperties`, which prevent the data stored in them from conflicting with that of other sprites. The variables are 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>configure SQL</strong></td>
<td>Configures SciSnap! for SQL access. For this, the global variables SQLData and SQLProperties are created and the initial properties are set. The sprite that executes the command will take the costume SQLDisconnected if it exists.</td>
</tr>
<tr>
<td><strong>is SQL configured?</strong></td>
<td>Returns &quot;true&quot; if the configuration is correct, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td><strong>connect to database server</strong></td>
<td>Connects to a database server whose address is in the script. This should be re-set, e.g. as localhost, if the default server is not used. If the connection attempt is successful, the executing sprite will take the costume SQLConnected if it is present.</td>
</tr>
<tr>
<td><strong>set SQLProperty</strong></td>
<td>Sets one of the properties in SQLProperties to the specified value.</td>
</tr>
<tr>
<td><strong>get SQLProperty</strong></td>
<td>Returns one of the properties in SQLProperties.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Import SQL-data from to SQLData</td>
<td>Imports a table into the SQLData data area. Example: Import of a query result.</td>
</tr>
<tr>
<td>read databases</td>
<td>Provides a list of currently available databases.</td>
</tr>
<tr>
<td>choose database no.</td>
<td>Selects one of the existing databases.</td>
</tr>
<tr>
<td>read tables</td>
<td>Returns a list of tables in the selected database.</td>
</tr>
<tr>
<td>attributes of table no.</td>
<td>Returns a list of attributes of the specified table.</td>
</tr>
<tr>
<td>exec SQL-command</td>
<td>Block for executing SQL statements to a database. The statements can be generated by select blocks or entered directly.</td>
</tr>
<tr>
<td>Predicates</td>
<td>Predicates to perform comparisons.</td>
</tr>
<tr>
<td>Predicates</td>
<td>Predicates to perform logical operations.</td>
</tr>
<tr>
<td>LIKE</td>
<td>Predicate for checking a string pattern.</td>
</tr>
<tr>
<td>IN ( )</td>
<td>Predicate to check if an element is included in items.</td>
</tr>
<tr>
<td>SUM ( )</td>
<td>Aggregate functions.</td>
</tr>
<tr>
<td>COUNT ( )</td>
<td></td>
</tr>
<tr>
<td>AVG ( )</td>
<td></td>
</tr>
<tr>
<td>MIN ( )</td>
<td></td>
</tr>
</tbody>
</table>
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.
### 3.4 The ImagePad-Library (SciSnap!ImagePadLibrary.xml)

*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. 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>configure thisSprite</code></td>
<td>Configures a sprite or the stage as <em>ImagePad</em>. The name of &quot;anotherSprite&quot; must be specified if required. The command must be executed once before working with a sprite as <em>ImagePad</em>. The target of the call takes a rectangular costume with the specified dimensions and colors.</td>
</tr>
<tr>
<td><code>is thisSprite an ImagePad?</code></td>
<td>Returns &quot;true&quot; if the configuration is correct, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td><code>imagePadProperty costumeProperties</code></td>
<td>Returns one of the properties in <em>myProperties</em>.</td>
</tr>
<tr>
<td><code>set ImagePadProperty costumeProperties</code></td>
<td>Sets one of the properties in <em>myProperties</em> to the specified value.</td>
</tr>
<tr>
<td><code>set ImagePad costume properties width: 400 height: 300 back color: 245 245 245 offsets: 0 0 on thisSprite</code></td>
<td>Sets the costume properties of the specified target.</td>
</tr>
<tr>
<td><code>set ImagePad line properties style: continuous width: 1 color: 0 0 0 fill color: 180 180 180 on thisSprite</code></td>
<td>Sets the line properties of the specified target.</td>
</tr>
<tr>
<td><code>draw line from 10 10 to 100 100 on thisSprite</code></td>
<td>elementary drawing operations</td>
</tr>
<tr>
<td><code>draw rectangle from 10 10 to 100 100 on thisSprite</code></td>
<td></td>
</tr>
<tr>
<td><code>draw circle center: 100 100 radius: 20 on thisSprite</code></td>
<td></td>
</tr>
<tr>
<td><code>draw text myText at 100 50 height: 12 horizontal? on thisSprite</code></td>
<td></td>
</tr>
<tr>
<td><code>draw list of points myData as circles size: 5 on thisSprite</code></td>
<td>Draws a list of &quot;points&quot; as circles or squares. Attention: JS-coordinates are used!</td>
</tr>
<tr>
<td><strong>import</strong> costume(RGB) data to myData on thisSprite</td>
<td>Imports image data.</td>
</tr>
<tr>
<td>add gray image of myData to ImagePad</td>
<td>Creates an image from the image data on a <em>ImagePad</em>.</td>
</tr>
<tr>
<td>affine transformation of costume currentCostume by</td>
<td>Returns the result of an affine transformation of a costume described by specifying three original points and three image points.</td>
</tr>
<tr>
<td>read image file with filepicker</td>
<td>Starts the file selection dialog.</td>
</tr>
<tr>
<td>set RGB at 1 1 on thisSprite to 255 100 30</td>
<td>Sets a pixel of the costume to the specified value.</td>
</tr>
<tr>
<td>RGB at 1 1 on thisSprite</td>
<td>Returns the value of a pixel of the costume.</td>
</tr>
<tr>
<td>set image value of myData at 1 1 on thisSprite to</td>
<td>Sets an image point in the data area to the specified value.</td>
</tr>
<tr>
<td>image value of myData at 1 1 on thisSprite</td>
<td>Returns the value of an image point from the data area.</td>
</tr>
<tr>
<td>image-value</td>
<td>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, and brightness values from a range.</td>
</tr>
<tr>
<td>brightness around 100 100 within radius 10 of myData of ImagePad</td>
<td>Returns the total brightness around a pixel in the specified radius.</td>
</tr>
<tr>
<td>draw list of points myData as circles radius: 5 on thisSprite</td>
<td>Draws a list of &quot;point&quot; as &quot;balls&quot;. Attention: JS-coordinates are used!</td>
</tr>
</tbody>
</table>
### 3.5 The PlotPad-Library (SciSnap!PlotPadLibrary.xml)

PlotPads are used to display graphs, histograms, etc. The current Sci-Snap!PlotPadLibrary was heavily designed by Rick Hessman, especially the PrettyPrinting and the line styles are from him. Many thanks for that!

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configure sprite as a PlotPad width: 400 height: 300 color: 245 245 245</td>
<td>Configures a sprite or the stage as a PlotPad. The name of &quot;anotherSprite&quot; must be specified if required. The command must be executed once before working with a sprite as a PlotPad. The target of the call takes a rectangular costume with the specified dimensions and colors.</td>
</tr>
<tr>
<td>is thisSprite a PlotPad?</td>
<td>Returns &quot;true&quot; if the configuration is correct, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>set PlotPadProperty costumeProperties</td>
<td>Sets one of the properties in myProperties to the specified value.</td>
</tr>
<tr>
<td>set PlotPad costume properties width: 400 height: 300 back color: 245 245 245 front color: 180 180 180 offsets: 0 0 on thisSprite</td>
<td>Sets the costume properties of the specified target.</td>
</tr>
<tr>
<td>set PlotPad line properties style: continuous</td>
<td>Sets the line properties of the specified target.</td>
</tr>
<tr>
<td>set PlotPad marker properties style: square connected? off color: 0 0 0 on thisSprite</td>
<td>Sets the marker (datapoint) properties of the specified target.</td>
</tr>
<tr>
<td>set PlotPad scale properties precision: 2 textheight: 12 number of intervals: 10 on thisSprite</td>
<td>Sets the scale properties of the specified target.</td>
</tr>
<tr>
<td>set PlotPad labels on thisSprite to title: Diagram Title titleheight: 18 x-label: x-label xLabelheight: 16 y-label: y-label yLabelheight: 16</td>
<td>Sets the label properties of the specified target.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>set PlotPad offsets from edges</strong> on thisSprite</td>
<td>Calculates the distances of the coordinate axes from the edges.</td>
</tr>
<tr>
<td><strong>set PlotPad ranges for x: -10 10 y: -10 10</strong> with border? on thisSprite</td>
<td>Sets the ranges of the axes of the coordinate system.</td>
</tr>
<tr>
<td><strong>set pretty ranges on PlotPad</strong> thisSprite</td>
<td>Sets the number ranges so that &quot;pretty&quot; labels are on the axes.</td>
</tr>
<tr>
<td><strong>pretty values for a PlotPad from -10 to 10 with 6 intervals</strong></td>
<td>Returns values for &quot;pretty&quot; labels of the axes.</td>
</tr>
<tr>
<td><strong>get ranges for PlotPad</strong> from myData with border 0.1</td>
<td>Redetermines the ranges, calculated from the data.</td>
</tr>
<tr>
<td><strong>ranges of 2-dim table</strong></td>
<td>Returns the ranges of a two-dimensional table.</td>
</tr>
<tr>
<td><strong>add graph</strong> ringified operator or polynomial to PlotPad thisSprite</td>
<td>Adds a function graph to the PlotPad, given as a list of polynomial coefficients or as a &quot;ringified&quot; term.</td>
</tr>
<tr>
<td><strong>add dataplot of numeric data</strong> myData to PlotPad thisSprite</td>
<td>Adds a data plot to the PlotPad.</td>
</tr>
<tr>
<td><strong>add dataplot of mixed data</strong> myData y-scale? x-scale? to PlotPad thisSprite</td>
<td>Adds a data plot for a two-dimensional data table to the PlotPad.</td>
</tr>
<tr>
<td><strong>add histogram of</strong> myData with 10 groups pretty formatted? to PlotPad thisSprite</td>
<td>Adds a histogram to the PlotPad.</td>
</tr>
<tr>
<td><strong>add axes and scales to PlotPad</strong> thisSprite</td>
<td>Adds axes and labels to the PlotPad.</td>
</tr>
<tr>
<td><strong>clear plot of</strong> thisSprite</td>
<td>Deletes the previous plots.</td>
</tr>
<tr>
<td><strong>convert value 100 to coordinate</strong> of PlotPad thisSprite</td>
<td>Returns the conversion of a numerical value into coordinates of the plotpad or the coordinate system used.</td>
</tr>
<tr>
<td><strong>SIMPLE PLOT</strong> of data:</td>
<td>Converts the mouse position into costume or graph coordinates.</td>
</tr>
<tr>
<td>x: 0 y: 0 width: 800 height: 400</td>
<td>Simple plot program for data.</td>
</tr>
</tbody>
</table>
3.6 The GraphPad-Library (SciSnap!GraphPadLibrary.xml)

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 SciSnap!GraphPadLibrary.

- **configure thisSprite as a GraphPad width: 400 height: 300 color: 245 245 245**
  
  Configures a sprite or the stage as a GraphPad. The name of "anotherSprite" must be specified if required. The command must be executed once before working with a sprite as GraphPad. The target of the call takes a rectangular costume with the specified dimensions and colors.

- **is thisSprite a GraphPad?**
  
  Returns "true" if the configuration is correct, otherwise "false".

- **set GraphPadProperty costumeProperties width: height: color:**
  
  Sets one of the properties in myProperties to the specified value.

- **set GraphPadProperty costumeProperties width: height: color:**
  
  Sets the costume properties of the specified target.

- **set GraphPad vertex properties growing? showsContent? color? directed? weighted?**
  
  Sets the vertex (node) properties of the specified target.

- **add n vertices to graph on thisSprite**
  
  Adds n vertices (nodes) to the graph at random positions.

- **new vertex at 100 100 content:**
  
  Adds one vertex (node) to the graph at the specified position.

- **move vertex 1 of graph on thisSprite to 100 100**
  
  Moves a vertex (node) to the specified position.

- **add 1 random edges to graph on thisSprite**
  
  Adds n random edges to the graph.

- **add edge from vertex 1 to vertex 2 to graph on thisSprite**
  
  Adds an edge to the graph between the named vertices (nodes).

- **draw graph on thisSprite**
  
  Draws the graph. Colors connected vertices (nodes) in the same color.

- **change weight of edge from vertex 1 to vertex 2 to 1 of graph on thisSprite**
  
  Changes the weight of an edge, if possible.
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight of edge from vertex 1 to vertex 2 of graph on thisSprite ▼</td>
<td>Returns the weight of an edge, if possible.</td>
</tr>
<tr>
<td>change content of vertex 1 of graph on thisSprite ▼</td>
<td>Changes the content of a vertex (node).</td>
</tr>
<tr>
<td>content of vertex 1 of graph on thisSprite ▼</td>
<td>Returns the content of a vertex (node).</td>
</tr>
<tr>
<td>ask for new weight of graph on thisSprite ▼</td>
<td>Asks for a new edge weight.</td>
</tr>
<tr>
<td>ask for new vertex content in graph on thisSprite ▼</td>
<td>Asks for a new vertex (node) content.</td>
</tr>
<tr>
<td>ask for new start vertex width of graph on thisSprite ▼</td>
<td>Asks for a new start vertex size.</td>
</tr>
<tr>
<td>delete vertex 1 of graph on thisSprite ▼</td>
<td>Deletes a vertex (node) of the graph.</td>
</tr>
<tr>
<td>delete edge from vertex 1 to vertex 2 of graph on thisSprite ▼</td>
<td>Deletes an edge of the graph.</td>
</tr>
<tr>
<td>set marker of vertex 1 of graph on thisSprite ▼</td>
<td>Marks a vertex (node).</td>
</tr>
<tr>
<td>remove marker of vertex 1 of graph on thisSprite ▼</td>
<td>Deletes the marking of a vertex (node).</td>
</tr>
<tr>
<td>remove all markers of graph on thisSprite ▼</td>
<td>Deletes all markers in the graph.</td>
</tr>
<tr>
<td>depth first search of content 1 starting at vertex 1 of graph on thisSprite ▼</td>
<td>Depth first search for a content starting from the vertex with the specified number.</td>
</tr>
<tr>
<td>breadth first search of content 1 starting at vertex 1 of graph on thisSprite ▼</td>
<td>Breadth first search for a content starting from the vertex with the specified number.</td>
</tr>
<tr>
<td>distance on thisSprite ▼ from vertex 1 to vertex 2</td>
<td>Spatial distance between two vertices on the sprite.</td>
</tr>
<tr>
<td>shortest path in graph from vertex 1 to vertex 2 on thisSprite ▼</td>
<td>Shortest path between two vertices in the graph.</td>
</tr>
<tr>
<td>list of all shortest paths in graph from vertex 1 to all connected vertices of graph on thisSprite ▼</td>
<td>List of all shortest paths from a vertex to all connected vertices in the graph.</td>
</tr>
<tr>
<td>vertex number at 100 50 of graph on thisSprite ▼</td>
<td>Number of a vertex at the given position on the sprite.</td>
</tr>
<tr>
<td>point 0 0 on stage ➞ point on graph thisSprite ▼</td>
<td>Converts stage coordinates to graph (JS-style) coordinates.</td>
</tr>
<tr>
<td>vertex number of Peter in graph of thisSprite ▼</td>
<td>Returns the number of the vertex with the specified content.</td>
</tr>
</tbody>
</table>

```javascript
configure thisSprite ▼ as a GraphPad
width: 400
height: 300
color: 245 245 245
add 100 vertices to graph on thisSprite ▼
add 100 random edges to graph on thisSprite ▼
```
### 3.7 The NeuralNetPad-Library (SciSnap!NNPadLibrary.xml)

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configure</td>
<td>Configures a sprite or the stage as NNPad. The name of &quot;anotherSprite&quot; must be specified if required. The command must be executed once before working with a sprite as NNPad. The target of the call takes a rectangular costume with the specified dimensions and colors.</td>
</tr>
<tr>
<td>is thisSprite</td>
<td>Returns &quot;true&quot; if the configuration is correct, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>set NNPadProperty</td>
<td>Sets one of the properties in myProperties to the specified value.</td>
</tr>
<tr>
<td>NNPadProperty</td>
<td>Returns one of the properties in myProperties.</td>
</tr>
<tr>
<td>set NNPad costume properties</td>
<td>Sets the costume properties of the specified target.</td>
</tr>
<tr>
<td>set NNPad properties</td>
<td>Sets the layer properties of the specified target.</td>
</tr>
<tr>
<td>NN add new weights</td>
<td>Inserts random weights for an NN of the specified width and depth.</td>
</tr>
<tr>
<td>NN output of</td>
<td>Returns the output of the nth layer of the NN for the specified input vector.</td>
</tr>
<tr>
<td>NN show status with input</td>
<td>Displays the status of the network for the specified input vector in color-coded form.</td>
</tr>
<tr>
<td>teach NN with input and target output</td>
<td>Trains the NN for the given input vector to achieve the given output vector.</td>
</tr>
</tbody>
</table>
4 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 watcher. 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. If you want to do the same programmatically, use the reporter block 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.
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/SQLLibrary 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 SQLProperties.

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 <read file with 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!
Example: Data import with the 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).

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]).

We get the total brightness and the number of measured pixels.
The export of data can again 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.
5 Examples and Tasks

5.1 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, Hilberto’s contribution must of course be adequately acknowledged by his co-representation.
5.2 Affine transformation of a triangle in $\mathbb{R}^2$

We define a triangle by its location vectors. Then we define three points in the plane and three points to map them to:

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 rather something for physics, Alberto presents the result.

```plaintext
configure sprite theSprite as MathPad
  width: 500
  height: 400
  color: 255 245 245
  dimension: 2
  maxValue: 10
  startPoint: 0 0

set MathPad properties lineWidth: 2
  onlyPoints: 0

plot object-of triangle on MathPad
  color: 255 0 0
  change startPoint? 0

set object-of affine transformation of triangle by sourcePoints --> targetPoints for MathPad on MathPad
  color: 0 0 255
  change startPoint? 0
```
5.3 Rotation of a pyramid in $\mathbb{R}^3$

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.

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.

So in total:
5.4 Graph of normal distribution

Using the *PlotPad*, the graph of the normal distribution is drawn.

![Graph of normal distribution](image_url)

```plaintext
configure thisSprite as a PlotPad width: 400 height: 300 color: 245 245 245
set PlotPad labelon thisSprite to title: NormalDistribution height: 18
x-label: x xLabelHeight: 16
y-label: \( \mu(0,1) \) yLabelHeight: 16
set PlotPad ranges for x: -5 5 y: 0 0.5 with border? \( \text{of} \) 0.11 pretty formatted? \( \text{of} \) thisSprite
add graph \( n \( x=0 \mu=0 \sigma=1 \) \) to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```
5.5 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 Carllsen }
set ages  to set of { 22 < and 65 > }
set professions to set of { teacher bricklayer retailer lawyer }
```

After that we generate four values from the "allowed" age range:

```
set actualAges to set of { }
set i to 0
repeat until  i = 4
  set age to pick random 1 to 100
  if age ε ages?
    set actualAges to actualAges u set of { age }
```

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

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Peterson</td>
<td>64</td>
</tr>
<tr>
<td>2</td>
<td>Peterson</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>Peterson</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>Peterson</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>Peterson</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>Peterson</td>
<td>46</td>
</tr>
<tr>
<td>7</td>
<td>Peterson</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>Peterson</td>
<td>46</td>
</tr>
<tr>
<td>9</td>
<td>Peterson</td>
<td>46</td>
</tr>
<tr>
<td>10</td>
<td>Peterson</td>
<td>44</td>
</tr>
<tr>
<td>11</td>
<td>Peterson</td>
<td>44</td>
</tr>
<tr>
<td>12</td>
<td>Peterson</td>
<td>44</td>
</tr>
<tr>
<td>13</td>
<td>Peterson</td>
<td>34</td>
</tr>
</tbody>
</table>
```

And thus nothing stands in the way of a transition to the topic of "relational databases", for example.
5.6 Representation of a set of points and the regression line

Using the SciSnap!DataLibrary, we create 100 random points scattering around a straight line with slope $m=0.5$ and intercept $b=0$. We plot the obtained points in a diagram.

In addition, the regression line is also drawn.
5.7 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:

After that we configure the current sprite to the PlotPad and draw the point set.

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

And last we add the interpolation polynomial.
**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-holistic 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.
5.8 Approximation of a tangent by secants

We want to show that a sequence of secant slopes converges against the slope 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:

We may as well draw it in.

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

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

... und die Sekanten in Farbabstufungen zeichnen.
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.
5.9 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 \( \pi \). In a compendium of formulas \(^{18}\) or on Wikipedia \(^{19}\) we can find a formula for calculating \( \pi \):

the Leibniz-series:

\[
\frac{\pi}{4} = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)}
\]

We can implement them directly in SciSnap!

But when is the result actually "good"?

To answer this question we create a table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.23215809405594</td>
</tr>
<tr>
<td>2</td>
<td>3.151490401079914</td>
</tr>
<tr>
<td>3</td>
<td>3.1425916543939544</td>
</tr>
<tr>
<td>4</td>
<td>3.1416926435905346</td>
</tr>
<tr>
<td>5</td>
<td>3.1416026534897203</td>
</tr>
<tr>
<td>6</td>
<td>3.1415936655867784</td>
</tr>
<tr>
<td>7</td>
<td>3.14159273535897814</td>
</tr>
</tbody>
</table>

With 10 million summands you have to wait a bit for the result!

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 \( \pi \).

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.

---

\(^{18}\) Ask your grandpa what it is. 😊

\(^{19}\) https://en.wikipedia.org/wiki/Leibniz_formula_for_\( \pi \)
We therefore have the exact *Scheme numbers* converted to inexact floating point representation before we enter them into the table.

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.
5.10 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 $F$ accelerating the ball on the circular path as $F = -G \cdot \sin \varphi = -m \cdot g \cdot \sin \varphi$. 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.

![Diagram](image1)

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

![Diagram](image2)

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 \( \varphi_{\text{Approx}} \) and the corresponding angular velocity. The "real" angle \( \varphi \) 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.

![Diagram](image3)

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.

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

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.
5.10 Fourier expansion for a square wave signal with numerical integration

One way to Fourier represent a $2\pi$-periodic function $f(x)$ is

$$f(x) \approx a_0 + \sum_{k=1}^{\infty} \left( a_k \cdot \cos(k \cdot x) + b_k \cdot \sin(k \cdot x) \right)$$

$$a_0 = \frac{1}{2\pi} \int_0^{2\pi} f(x) \cdot dx$$

$$a_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \cdot \cos(k \cdot x) \cdot dx$$

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

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 $2\pi$. A function that returns this would be e.g.

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

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

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.

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

For the other coefficients we have to complete the trigonometric terms and obtain for
These terms now only have to be summed up:
This gives us the following script in total:

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 judged 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.

<table>
<thead>
<tr>
<th>( n ) summands</th>
<th>( i ) intervals</th>
<th>result</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>100</td>
<td></td>
<td>Actually quite good except for &quot;slips&quot; that are at the jump points of the function.</td>
</tr>
</tbody>
</table>
50

That should be clear.

Better, but only slightly better than with 100 summands.

Worse, but only marginally worse than with 50 summands.

Perhaps quite a good compromise between accuracy and computation time requirements.

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.
5.11 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 quite a size, which we load “in one swoop” into the SciSnap! data area, the SciSnap!Data variable. We get almost 600,000 records.

Their column headers we split off from the table.

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.

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

That’s what we thought!

And what about the correlation? First, we throw out 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.

And what does this number want to tell us now??? We don’t know - but we can read up and learn! 😊
5.12 Income data from the US Census Income Dataset

We want to dig into some data and therefore download the Census Income Dataset from the net. The corresponding CSV file can be loaded from the file directory in SciSnapData and displayed immediately. It contains 32562 records. Double-clicking or right-clicking on it and selecting "open in dialog..." shows all columns.

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.

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:

---

20 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.

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.

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

And what does that mean?

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.
5.13 New York Citibike Tripdata 2: data processing

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.

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:

The average duration of borrowing, in terms of gender:

That’s what we thought!

Let’s see if they are lazier on Broadway:

I see. Probably Central Park is even worse!

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?
5.14 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 (example 48), 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.

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.
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.
5.15 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 stations ²¹.

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

---

²¹ This will take some time!
At least in midtown Manhattan, I don't think we have to worry about finding a borrowing station!

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.

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

Lending:

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.
5.16 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`.

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

```
set data to read file with filepicker

starname HD 116608

data

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>351.00</td>
<td>8.1860e-13</td>
<td>0.3586</td>
</tr>
<tr>
<td>2</td>
<td>351.14</td>
<td>8.1770e-13</td>
<td>0.3684</td>
</tr>
<tr>
<td>3</td>
<td>351.28</td>
<td>8.3890e-13</td>
<td>0.3680</td>
</tr>
<tr>
<td>4</td>
<td>351.42</td>
<td>8.4400e-13</td>
<td>0.3704</td>
</tr>
<tr>
<td>5</td>
<td>351.56</td>
<td>8.3100e-13</td>
<td>0.3649</td>
</tr>
<tr>
<td>6</td>
<td>351.70</td>
<td>8.3270e-13</td>
<td>0.3659</td>
</tr>
<tr>
<td>7</td>
<td>351.84</td>
<td>8.3740e-13</td>
<td>0.3682</td>
</tr>
<tr>
<td>8</td>
<td>351.98</td>
<td>8.3200e-13</td>
<td>0.3661</td>
</tr>
<tr>
<td>9</td>
<td>352.12</td>
<td>8.0700e-13</td>
<td>0.3655</td>
</tr>
<tr>
<td>10</td>
<td>352.26</td>
<td>8.2450e-13</td>
<td>0.3405</td>
</tr>
<tr>
<td>11</td>
<td>352.40</td>
<td>8.3900e-13</td>
<td>0.3363</td>
</tr>
<tr>
<td>12</td>
<td>352.54</td>
<td>8.3350e-13</td>
<td>0.3354</td>
</tr>
</tbody>
</table>

normalizedData

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>351.00</td>
<td>8.86553745</td>
</tr>
<tr>
<td>2</td>
<td>351.14</td>
<td>8.89551766</td>
</tr>
<tr>
<td>3</td>
<td>351.28</td>
<td>9.18770176</td>
</tr>
<tr>
<td>4</td>
<td>351.42</td>
<td>9.24535746</td>
</tr>
<tr>
<td>5</td>
<td>351.56</td>
<td>9.10198033</td>
</tr>
<tr>
<td>6</td>
<td>351.70</td>
<td>9.11979881</td>
</tr>
<tr>
<td>7</td>
<td>351.84</td>
<td>9.17172366</td>
</tr>
<tr>
<td>8</td>
<td>351.98</td>
<td>9.19123241</td>
</tr>
<tr>
<td>9</td>
<td>352.12</td>
<td>9.88440165</td>
</tr>
<tr>
<td>10</td>
<td>352.26</td>
<td>9.85919085</td>
</tr>
<tr>
<td>11</td>
<td>352.40</td>
<td>9.83554330</td>
</tr>
<tr>
<td>12</td>
<td>352.54</td>
<td>9.83775633</td>
</tr>
</tbody>
</table>
```

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

```
show spectrum

configure PlotPad as a PlotPad with width: 500, height: 400, color: 245, 245, 245
set PlotPad labels on PlotPad to starname
set PlotPad label for y-axis: normalized flux
set PlotPad label for x-axis: wavelength / nm
set PlotPad ranges for x: 300 to 800, y: 0 to 3.5 with tickloc: 0 of 0.5 pretty format: 7
set PlotPad marker properties style: 6, width: 5, color: 0, 0, connected: 0
add dataplot of numeric data: normalizedData to PlotPad PlotPad
add axes and scales to PlotPad
```
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 $\text{compressionRate}$ and complete the script before the diagram generation.

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.
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.

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.
5.17 Classification with the kNN method

In the Hertzsprung-Russel-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-Russel-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.
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?
5.21 Drawing a function and its derivatives

We want to use a SciSnapPlotPad 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.

```plaintext
configure thisSprite as a PlotPad width: 400 height: 400 color: 245 245 245
set PlotPad labels on thisSprite to title: FunctionAndDerivatives titleHeight: 18 x-label: x xLabelHeight: 16 y-label: y yLabelHeight: 16
set pretty ranges on PlotPad thisSprite
set PlotPad line properties style: continuous width: 2 color: 0 0 0 on thisSprite
add graph 0.1 × × × × − − − 0 × × × × to PlotPad thisSprite
set PlotPad line properties style: dashed width: 2 color: 255 0 0 0 on thisSprite
add graph [3, 0, 0, 0] to PlotPad thisSprite
set PlotPad line properties style: dot-dot width: 2 color: 0 255 0 0 on thisSprite
add graph [0.6, 0, 0, 0] to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```

**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.
5.22 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.

```python
import takeCSVdata from

20 random points near \( f(x) = x^3 - x \) to SciSnapData

set SciSnapData to SciSnapData sorted by column 1

configures

| height: 200 | color: 245 245 245 |

get ranges for PlotPad thisSprite

from SciSnapData with border 0.07

set pretty ranges on PlotPad thisSprite

set PlotPad labels on thisSprite
title: RandomData
titleheight: 40
x-label: \( f(x) \) on thisSprite
y-label: \( y \) on thisSprite

set PlotPad line properties style: rainbow
width: 3 color: 00 00 00 on thisSprite

set PlotPad marker properties style: square width: 6

add dataplot of numeric data SciSnapData to PlotPad thisSprite

add axes and scales to PlotPad thisSprite
```
5.23 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.
5.24 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.

Let's first get an overview of this:

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

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

... and fit the graph: We show the semi-logarithmically plotted data and the regression lines for the two halves of the data.
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.
5.25 *Shadow lengths in the lunar crater Tycho*

We create an image of the lunar crater on an *ImagePad*, import its data and create a section 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*. 
5.26 Plot of mixed data

Text data is often combined with numerical data. An example would be the sales data of different representatives in one year in one 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.
5.27 A simple SQL query

As an example, let's ask for 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 kurs WHERE kursnummer LIKE '%Eng%'
to SQLData
```

5.28 A more complex SQL query

Query 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 Name, AVG(punkte) FROM Schueler Kurs WHERE schueler.ID_nummer = Kurs.ID_nummer AND Kurs.kursnummer LIKE '%Eng%'
GROUP BY Name
HAVING AVG(punkte) <= DESC
LIMIT 10
to SQLData
```
5.30 Dealing with the SQL library

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

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

Example: For statistical purposes, the schueler table should be searchable by different criteria.
5.31 Simple random graphic

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

<table>
<thead>
<tr>
<th>for</th>
<th>i</th>
<th>1 to 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>typ</td>
<td>to</td>
</tr>
<tr>
<td>set</td>
<td>x</td>
<td>to</td>
</tr>
<tr>
<td>set</td>
<td>y</td>
<td>to</td>
</tr>
<tr>
<td>set</td>
<td>typ</td>
<td>to</td>
</tr>
<tr>
<td>set</td>
<td>ImagePad line properties style</td>
<td>to</td>
</tr>
<tr>
<td>fill color</td>
<td>r</td>
<td>g</td>
</tr>
<tr>
<td>if</td>
<td>typ</td>
<td>=</td>
</tr>
<tr>
<td>draw line from</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>pick random</td>
<td>1</td>
<td>to</td>
</tr>
<tr>
<td>to</td>
<td>on ImagePad</td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>typ</td>
<td>=</td>
</tr>
<tr>
<td>draw rectangle from</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>pick random</td>
<td>1</td>
<td>to</td>
</tr>
<tr>
<td>pick random</td>
<td>1</td>
<td>to</td>
</tr>
<tr>
<td>if</td>
<td>typ</td>
<td>=</td>
</tr>
<tr>
<td>fill rectangle from</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>on ImagePad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>typ</td>
<td>=</td>
</tr>
<tr>
<td>draw circle centers</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>radius</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>on ImagePad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>typ</td>
<td>=</td>
</tr>
<tr>
<td>fill circle centers</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>radius</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>on ImagePad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>if</td>
<td>text</td>
<td>Hello</td>
</tr>
<tr>
<td>height</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>horizontal?</td>
<td>0</td>
<td>on ImagePad</td>
</tr>
<tr>
<td>else</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>Hello</td>
<td>at</td>
</tr>
<tr>
<td>height</td>
<td>pick random</td>
<td>1</td>
</tr>
<tr>
<td>horizontal?</td>
<td>0</td>
<td>on ImagePad</td>
</tr>
</tbody>
</table>

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.
5.32 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.

```
configure thisSprite as an ImagePad width: 400 height: 300 color: 245 245 245
import FITSData from read image file with filepicker
to myData on thisSprite
add false-color image of myData to ImagePad
min/max: 0 32000 log? ✓ on thisSprite
```

5.33 Section through an image of the lunar crater Tycho

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

```
import costume(RGB) data from currentCostume
to myData on thisSprite
set data to slice-data on thisSprite by mouse
```
5.36 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 RBG values.

We convert these into gray values.

We switch to the PlotSprite and copy the loaded data of the DataSprite.

Now we can plot the data as a histogram e.g. on a new PlotPad.

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.
5.37 Simulation of a planetary transit in front of the sun

We look for a nice picture of the sun (source here: [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 \((x=image\ width+2r)\) 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 exoplanets.
5.38 Affine transformation of an image

In the SciSnap!ImagePadLibrary 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 duplicate of the ImagePad and ask it to display the transformed image as a costume.
5.39 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 SciSnapDataLibrary 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. 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!
5.40 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.

5.41 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.
5.42 "Edges per vertex" histogram in a random graph

5.43 "Edges per vertex" histogram in a scalefree graph
5.44 Breadth and depth search in the 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.

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

**Tasks:**

1. Determine over how many 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 such a program would be useful to use.

   c: Decision trees play a role in certain applications of machine learning (Decision Tree Classification). Learn about the method and its applications.
5.45 A simple perceptron as a graph

We want to ask Hilberto to use a GraphPad to illustrate how a simple Perceptron 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:

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.

The output of the network is determined by multiplying the values of the input neurons by the corresponding edge weights and summing the results. The result is then 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.

https://en.wikipedia.org/wiki/Perceptron
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.

\(\smiley\)

**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.
5.46 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.

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!
5.47 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:

Training states after every 20 training steps:

The output of the net after training:

The positions of the largest or smallest value of the output can be determined directly using the SciSnap!DataLibrary:
5.48 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. 2x10^7 pixels. The input layer consists of 3x2x10^7 MB numerical values between 0 and 255 (if we omit the transparency byte). To the next layer there are then (6x10^7)^2=3.6x10^{15} connections - and then two more times. In total, 3x3.6x10^{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) on the basis of 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 nxn matrices. NeuralNetPads should therefore master linear algebra. The only new feature is the softmax function 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 the width 3 and the depth 2.

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.
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.

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 10000 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 30000 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: 12x2) 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, it must import the image data. Then it 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>.

The color values of the reduced image are assembled by Alberto into an input vector. These are then modeled with the softmax function of the DataLibrary 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 about 100 training runs with a higher learning rate and another 100 with a lower one for fine-tuning, we achieve recognition rates of 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.
5.49 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 a DataSprite for auxiliary operations, an ImageSprite for the actual image and - of course - a NeuralNetSprite 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.

<table>
<thead>
<tr>
<th>initialize</th>
<th>train the net</th>
<th>next costume</th>
<th>stop learning</th>
<th>draw digit (0/1)</th>
</tr>
</thead>
</table>

current digit

<table>
<thead>
<tr>
<th>convolution:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>reLU:</td>
<td></td>
</tr>
<tr>
<td>pooling (stride:4):</td>
<td></td>
</tr>
</tbody>
</table>

final feature maps:

means:

recognized digit: Ø

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 image above, after two stages of reduction, four feature maps of 16x16 pixels each are left, each of which has been run twice through the operations Convolution reLU MaxPooling: 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.
In the image above, after two stages of reduction, four feature maps of 16x16 pixels each are left, each of which has been run twice through the operations Convolution $\rightarrow$ reLU $\rightarrow$ Max-Pooling: 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 then applies a convolution with two different ones. 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 \( \text{output with } \langle \text{input} \rangle \). 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 - has the network learned anything? Let's write a number:

Well - there is still room for improvement!
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.
5.51 k-means-Clustering

In the data palette, two blocks are available for k-means clustering, but they (of course) only provide the final result. Let's illustrate the whole process here. For this we create a set of e.g. 100 random points with "JavaScript coordinates", to each of 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. We determine the colors, initially gray, from the cluster number.

We number the centers and enter their coordinates in a list center-Paths so that we can track their movements ...

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

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

The procedure is continued until there are no more changes in the clusters. We plot the movements of the centers using the stored positions.
Weg get the result:

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

```plaintext
configure theStage as an ImagePad width: 400 height: 300 color: 245 245 245
set data to "means clustering for 100 new random points"
show data as center
```
5.52 DNA relatedness and Levenshtein distance

We want to determine the closest "relative" to a given DNA sequence from a list of DNA sequences, e.g. to determine whether an animal was killed by a wolf. To do this, we first generate a DNA list.

Using the Levenshtein distance between two strings, we can now determine which of the strings is closest to the given one and how large their distance is.
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!
# List of examples

In most examples blocks of several libraries are used together!

<table>
<thead>
<tr>
<th>Mathematics</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Representation of complex numbers</td>
<td>42</td>
</tr>
<tr>
<td>2. Affine transformation of a triangle in $\mathbb{R}^2$</td>
<td>43</td>
</tr>
<tr>
<td>3. Rotation of a pyramid in $\mathbb{R}^3$</td>
<td>44</td>
</tr>
<tr>
<td>4. Graph of normal distribution</td>
<td>45</td>
</tr>
<tr>
<td>5. Cartesian product of three sets</td>
<td>46</td>
</tr>
<tr>
<td>6. Representation of a set of points and the regression line</td>
<td>47</td>
</tr>
<tr>
<td>7. Interpolation polynomial through $n$ points</td>
<td>48</td>
</tr>
<tr>
<td>8. Approximation of a tangent by secants</td>
<td>50</td>
</tr>
<tr>
<td>9. Finite series</td>
<td>52</td>
</tr>
<tr>
<td>10. Application of the Taylor series to the mathematical pendulum</td>
<td>54</td>
</tr>
<tr>
<td>11. Fourier expansion for a square wave signal with numerical integration</td>
<td>57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>12. NY Citibike Tripdata 1: Correlations</td>
<td>61</td>
</tr>
<tr>
<td>13. Income data from the US Census Income Dataset</td>
<td>62</td>
</tr>
<tr>
<td>14. NY Citibike Tripdata 2: data processing</td>
<td>64</td>
</tr>
<tr>
<td>15. Under- and overfitting</td>
<td>65</td>
</tr>
<tr>
<td>16. NY Citibike Tripdata 3: World Map Library</td>
<td>68</td>
</tr>
<tr>
<td>17. Star spectra</td>
<td>72</td>
</tr>
<tr>
<td>18. Classification in the HR diagram according to the kNN method</td>
<td>75</td>
</tr>
<tr>
<td>19. Data import and export: CSV import</td>
<td>38</td>
</tr>
<tr>
<td>20. Data import and export: JSON import</td>
<td>39</td>
</tr>
<tr>
<td>21. Data import and export: writing CSV and text data to a file</td>
<td>41</td>
</tr>
<tr>
<td>22. k-means-Clustering</td>
<td>112</td>
</tr>
<tr>
<td>23. DNA-Relatedness und Levenshtein-Distance</td>
<td>115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Diagrams</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>24. Drawing a function and its derivatives</td>
<td>77</td>
</tr>
<tr>
<td>25. Data plot of random data scattering around a function graph</td>
<td>78</td>
</tr>
<tr>
<td>26. Histogram of random values</td>
<td>79</td>
</tr>
<tr>
<td>27. Covid-19 data analysis</td>
<td>80</td>
</tr>
<tr>
<td>28. Shadow lengths in the lunar crater Tycho</td>
<td>82</td>
</tr>
<tr>
<td>29. Plot of mixed data</td>
<td>83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SQL</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>30. Simple SQL query</td>
<td>84</td>
</tr>
<tr>
<td>31. More complex SQL query</td>
<td>84</td>
</tr>
<tr>
<td>32. Data import and export: SQL import</td>
<td>39</td>
</tr>
<tr>
<td>33. Dealing with the SQL library</td>
<td>85</td>
</tr>
</tbody>
</table>
### Image processing

<table>
<thead>
<tr>
<th>Example</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>34. Random graphic</td>
<td>86</td>
</tr>
<tr>
<td>35. False color image of a lunar crater</td>
<td>87</td>
</tr>
<tr>
<td>36. Slice through the image of the lunar crater Tycho</td>
<td>87</td>
</tr>
<tr>
<td>37. Data import and export: false color image of Saturn</td>
<td>38</td>
</tr>
<tr>
<td>38. Data import and export: data import with the mouse</td>
<td>40</td>
</tr>
<tr>
<td>39. Plot of image data as histogram</td>
<td>88</td>
</tr>
<tr>
<td>40. Simulation of a planetary transit in front of a sun</td>
<td>89</td>
</tr>
<tr>
<td>41. Affine transformation of an image</td>
<td>90</td>
</tr>
<tr>
<td>42. Kernel applications for edge detection in images</td>
<td>91</td>
</tr>
</tbody>
</table>

### Graphs

<table>
<thead>
<tr>
<th>Example</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>43. Mean distances in a random graph (Small Worlds)</td>
<td>92</td>
</tr>
<tr>
<td>44. Mean distances in a scalefree graph (Small Worlds)</td>
<td>92</td>
</tr>
<tr>
<td>45. Histogram &quot;edges per vertex&quot; in a random graph</td>
<td>93</td>
</tr>
<tr>
<td>46. Histogram &quot;Edges per vertex&quot; in a scalefree graph</td>
<td>93</td>
</tr>
<tr>
<td>47. Breadth and depth search in the family tree</td>
<td>94</td>
</tr>
<tr>
<td>48. A simple perceptron as a graph</td>
<td>96</td>
</tr>
<tr>
<td>49. A simple learning perceptron as a graph</td>
<td>98</td>
</tr>
</tbody>
</table>

### Neural networks

<table>
<thead>
<tr>
<th>Example</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>50. Training of a neural network</td>
<td>100</td>
</tr>
<tr>
<td>51. Traffic sign recognition with a neural network</td>
<td>101</td>
</tr>
<tr>
<td>52. Character recognition with a Convolutional Neural Network CNN</td>
<td>106</td>
</tr>
</tbody>
</table>
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


[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


[NYcitibike] https://www.citibikenyc.com/system-data

[SchulAstro] www.schul-astronomie.de


[UniGOE] Institut für Astrophysik, Universitaet Goettingen