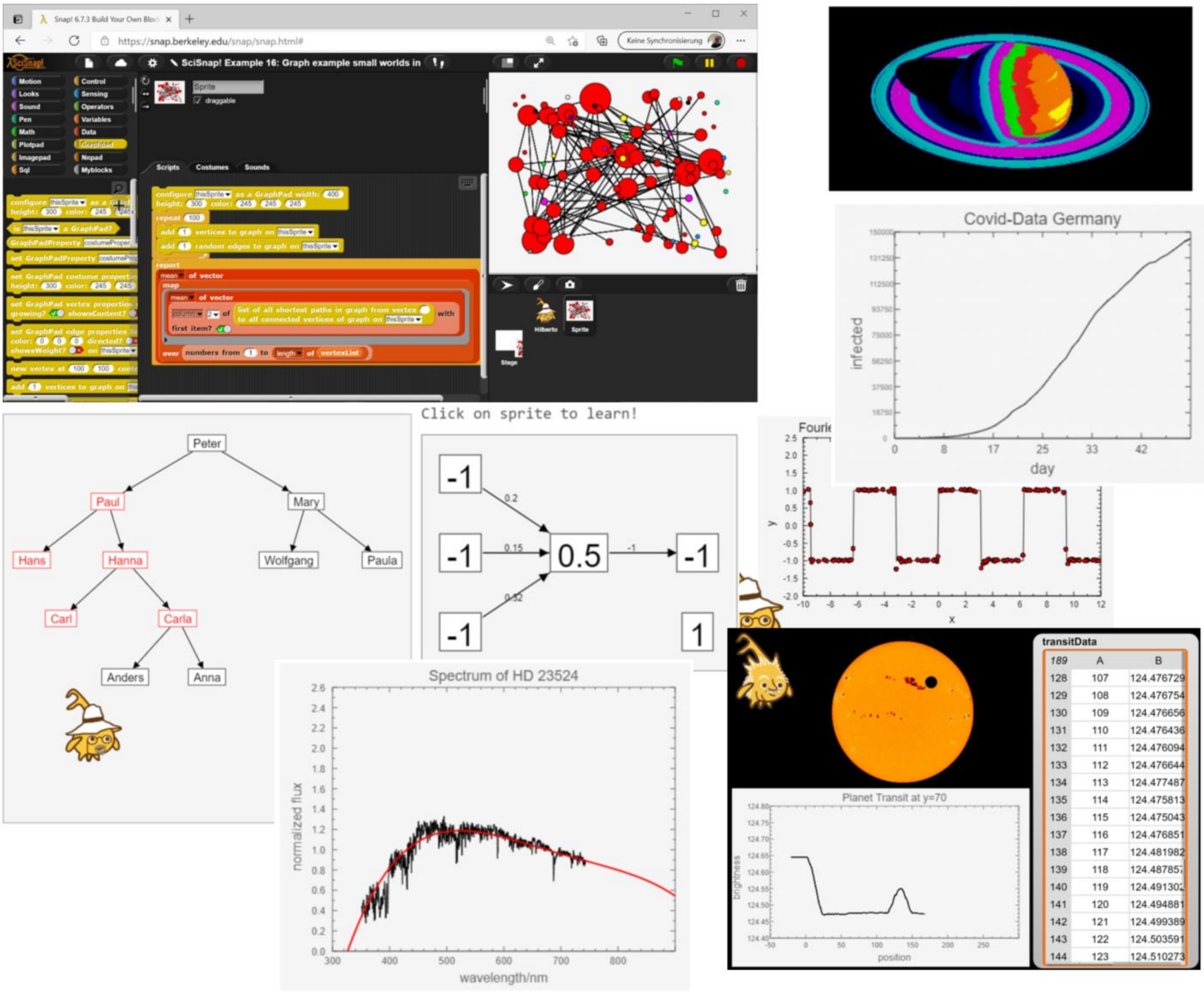
Eckart Modrow

Programming with SciSnap!



189	A	B
128	107	124.476729
129	108	124.476754
130	109	124.476656
131	110	124.476436
132	111	124.476094
133	112	124.476644
134	113	124.477487
135	114	124.475813
136	115	124.475043
137	116	124.476851
138	117	124.481982
139	118	124.48785
140	119	124.49130
141	120	124.494881
142	121	124.499389
143	122	124.503591
144	123	124.510273

© Eckart Modrow 2021

emodrow@informatik.uni-goettingen.de

This work is licensed under a Creative Commons attribution - Non-Commercial - Share Alike 4.0 International License. It allows download and redistribution of the complete work with mention of my name, but no editing or commercial use. 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

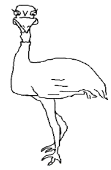
http://emu-online.de/SciSnap.zip or
http://emu-online.de/ProgrammingWithSciSnap.pdf

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

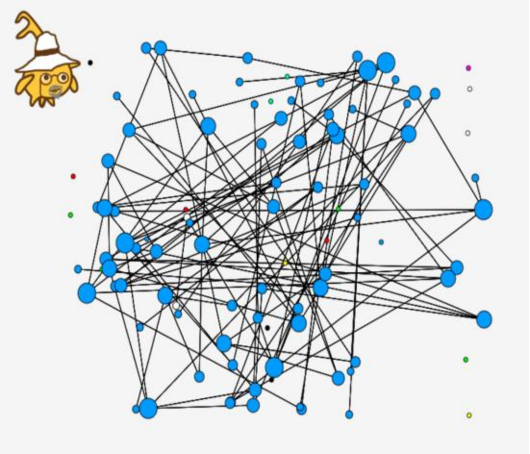
[1] https://scratch.mit.edu/

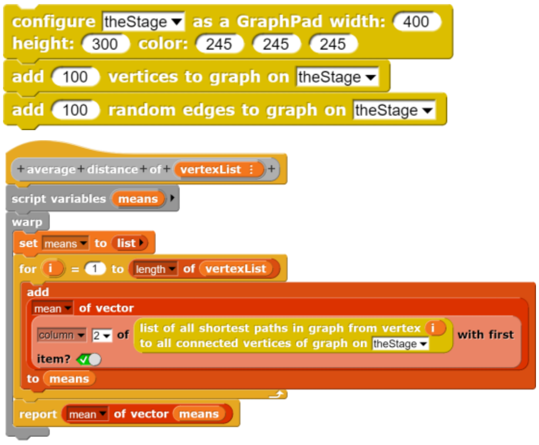
[2] https://snap.berkeley.edu/

[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.





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

```
+average+distance+of+ vertexList ⋮ +
script variables means ▸
warp
  set means ▾ to list ▸
  for i = 1 to length ▾ of vertexList
    add
      mean ▾ of vector
        column ▾ 2 ▾ of (list of all shortest paths in graph from vertex i
        to all connected vertices of graph on theStage ▾) with first
        item? ✓
    to means
  report mean ▾ of vector means
```

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

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*

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

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. [7] 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* [8] language*,* based on the "CS Bible" "Structure and Interpretation of Computer *Programs*" [9] 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* [10]*"* and the *Snap!*-fork *SQL-Snap!* [11]*.* 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&ProjectName=SciSnap!-Starter&editMode

[8] https://en.wikipedia.org/wiki/MIT/GNU_Scheme

[9] https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book.html

[10] as well as other materials on http://emu-online.de

[11] http://snapextensions.uni-goettingen.de/

with Snap!" [12] – and of course, in the *Snap!*-manual[13]. 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* [14] will take care of the more scientific applications, *Hilberto* [15] 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

[13] https://snap.berkeley.edu/snap/help/SnapManual.pdf

[14] he works in astrophysics with Rick Hessman

[15] https://de.wikipedia.org/wiki/David_Hilbert

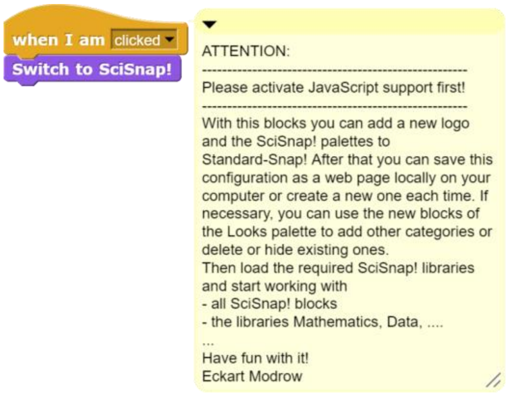
Content

1 SciSnap!-Starter

1.1 Create start configurations[16]

As long as *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[17]. Since I also want to work with the current *Snap!* version, more serious interventions in the actual code are not allowed. *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 *SciSnap!* applications:

1. New palettes and some global variables *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 *Snap!* project, e.g. in the cloud.



ATTENTION:

Please activate JavaScript support first!

With this blocks you can add a new logo and the SciSnap! palettes to Standard-Snap! After that you can save this configuration as a web page locally on your computer or create a new one each time. If necessary, you can use the new blocks of the Looks palette to add other categories or delete or hide existing ones.

Then load the required SciSnap! libraries and start working with

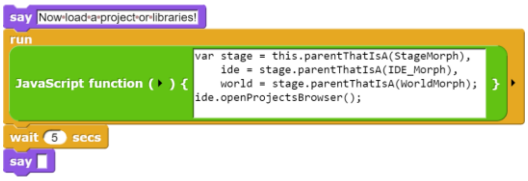
- all SciSnap! blocks
- the libraries Mathematics, Data, ...

...

Have fun with it!

Eckart Modrow

2. You should include a link to "your" SciSnap! starter project in the favorites bar in a form that executes the "green flag commands" immediately: e.g. as `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.


say Now load a project or libraries!
run
JavaScript function ( ) {
}
wait 5 secs
say

```
var stage = this.parentThatIsA(StageMorph),
    ide = stage.parentThatIsA(IDE_Morph),
    world = stage.parentThatIsA(WorldMorph);
ide.openProjectsBrowser();
```

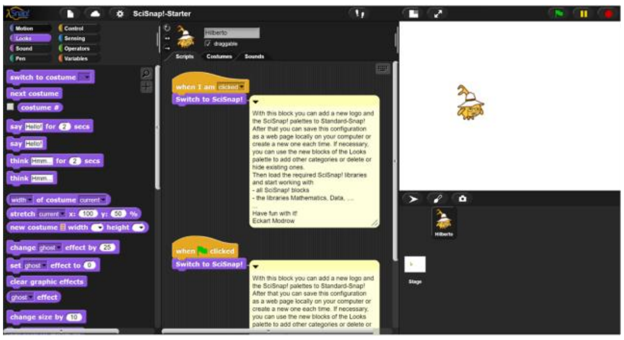
3. Work is done in this configuration and the current project is again stored in the cloud.
4. If you want to continue working on the current project later, then load the Starter project and with it the current *Snap!* version, e.g. with the link in the favorites bar. After its execution, the process 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 *SciSnap!* blocks have to be reloaded.

[16] Due to the current problems with JavaScript, JavaScript must first be enabled in the tools menu.

[17] To be clear, using the import library block, you can also load the SciSnap! libraries into any other palette. So you don't need to create new palettes!

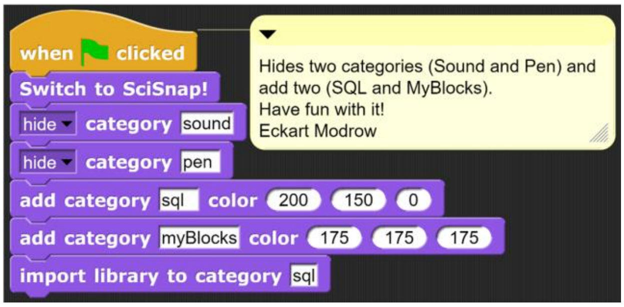
We will go through this with an example: The goal is to work with the SQL blocks and additionally insert a palette for your own blocks. The palettes for sound and the pen are not to be used.

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.



With this block you can add a new logo and the SciSnap! palettes to Standard-Snap! After that you can save this configuration as a web page locally on your computer or create a new one each time. If necessary, you can use the new blocks of the Looks palette to add other categories or delete or hide existing ones.
Then load the required SciSnap! libraries and start working with
- all SciSnap! blocks
- the libraries Mathematics, Data, ...
...
Have fun with it!
Eckart Modrow

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".

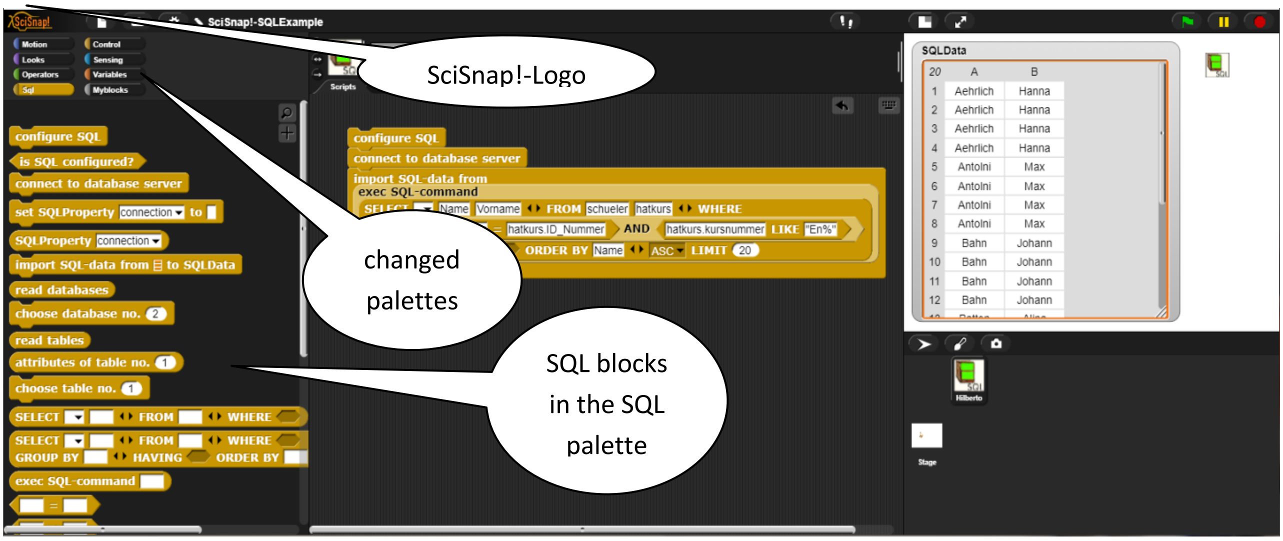


```
when ⚑ clicked
Switch to SciSnap!
hide category sound
hide category pen
add category sql color 200 150 0
add category myBlocks color 175 175 175
import library to category sql
```

Hides two categories (Sound and Pen) and add two (SQL and MyBlocks).
Have fun with it!
Eckart Modrow

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](http://snap.berkeley.edu/snap/snap.html#present:Username=emodrow&ProjectName=SciSnap!-SQLStarter&editMode)
After that it will be executed.

As a result we get a SQL working environment.



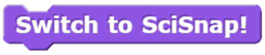
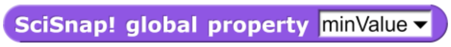
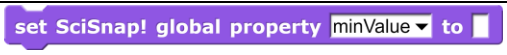
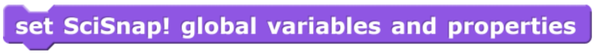
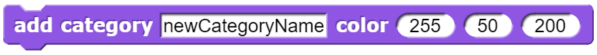
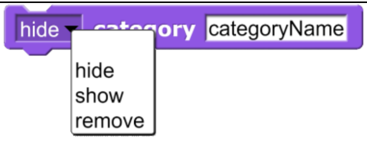
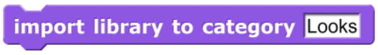
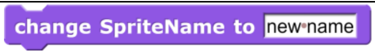
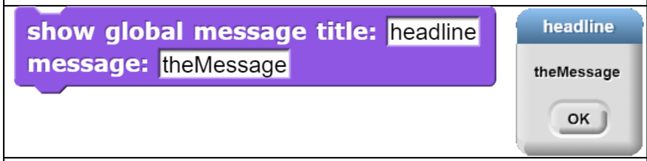
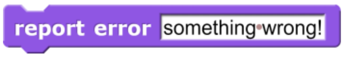
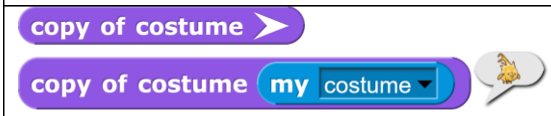
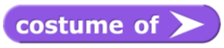
```
configure SQL
connect to database server
import SQL-data from
  exec SQL-command
    SELECT ▾ Name Vorname ◂▸ FROM schueler hatkurs ◂▸ WHERE
      ... = hatkurs.ID_Nummer AND hatkurs.kursnummer LIKE "En%"
      ORDER BY Name ◂▸ ASC ▾ LIMIT 20
```

SQLData

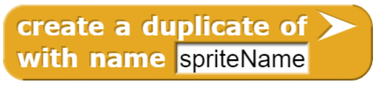
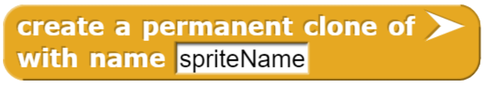
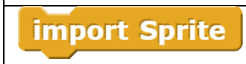
20	A	B
1	Aehrlich	Hanna
2	Aehrlich	Hanna
3	Aehrlich	Hanna
4	Aehrlich	Hanna
5	Antolni	Max
6	Antolni	Max
7	Antolni	Max
8	Antolni	Max
9	Bahn	Johann
10	Bahn	Johann
11	Bahn	Johann
12	Bahn	Johann

1.2 New blocks in the standard palettes of *Snap!*

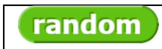
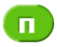
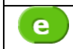
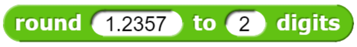
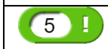
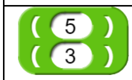
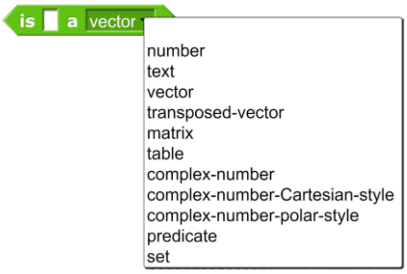
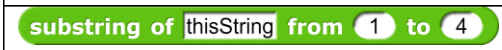
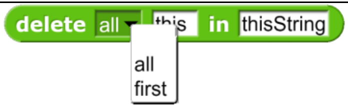
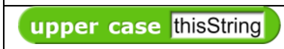
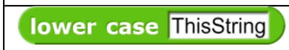
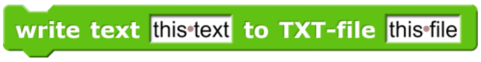
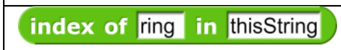
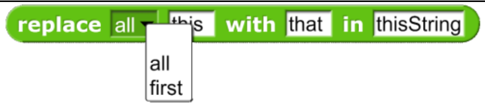
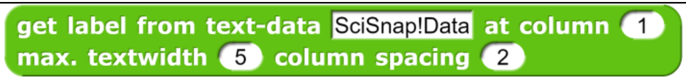
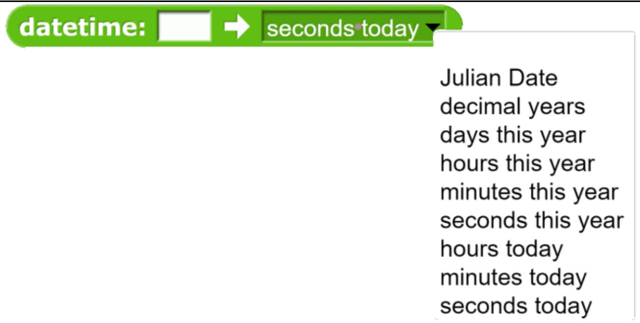
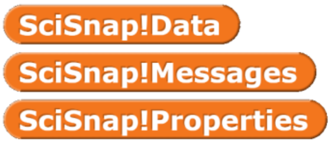
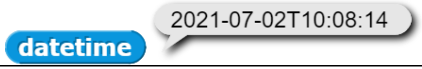
The additional blocks of the *Looks* palette are there because they have to be assigned to a standard *Snap!* palette to be loaded. The *Looks* palette is where they fit best and are the least disruptive. There are the following blocks:

 Switch to SciSnap!	Creates the *SciSnap!* logo and displays it, generates the desired palettes. Enlarges the stage to 800x600 pixels.
 SciSnap! global property minValue ▾	Reads a property.
 set SciSnap! global property minValue ▾ to	Allows you to modify or create a property.
 set SciSnap! global variables and properties	Creates three global *SciSnap!* variables "*SciSnap!Properties*", "*SciSnap!Data*" and "*SciSnap!Messages*". Sets some properties.
 add category newCategoryName color 255 50 200	Inserts a palette with the named name and RGB colors mentioned.
 hide ▾ category categoryName hide show remove	Hides, shows, or deletes a palette. Snap's standard palettes! will not be deleted.
 import library to category Looks	Imports a library of any name into the specified palette. The library is selected with the mouse.
 change SpriteName to new name	Changes the name of the current sprite.
 show global message title: headline message: theMessage headline theMessage OK	Creates a message window in the center of the screen.
 report error something wrong!	Outputs an error, if possible, via the current sprite and enters it into *SciSnap!Messages*.
 copy of costume copy of costume my costume ▾	Provides a copy of the current costume.
 costume of	Provides the costume of a sprite, e.g. for pooling operations.

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

 create a duplicate of with name spriteName	Creates a duplicate of the specified sprite with the specified name.
 create a permanent clone of with name spriteName	Creates a permanent clone of the specified sprite with the specified name.
 import Sprite	Imports a locally stored sprite.

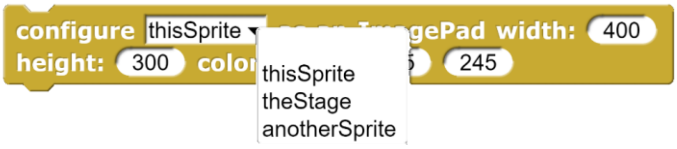
The *Operators* palette contains additional mathematical operators and string functions:

 random	Returns a random number between 0 and 1.
 π	Returns π.
 e	Returns the Euler number e.
 round 1.2357 to 2 digits	Returns the specified number rounded to the specified number of decimal places.
 5 !	Returns n!
 (5) (3)	Returns the binomial coefficient.
 is a vector number text vector transposed-vector matrix table complex-number complex-number-Cartesian-style complex-number-polar-style predicate set	Some type checks.
 substring of thisString from 1 to 4	Returns a partial string of strings.
 delete all this in thisString all first	Deletes all or the first string(s) in a string.
 upper case thisString	Returns a string in capital letters.
 lower case ThisString	Returns a string in lowercase.
 write text this text to TXT-file this file	Writes a string to a text file in the browser's download folder.
 index of ring in thisString	Returns the position of a substring.
 replace all this with that in thisString all first	Replaces all or the first string(s) in a string.
 get label from text-data SciSnap!Data at column 1 max. textwidth 5 column spacing 2	Reads a column heading from a table, e.g. to chart label.
 datetime: ➜ seconds today Julian Date decimal years days this year hours this year minutes this year seconds this year hours today minutes today seconds today	Conversions from *datetime* to the specified values.
In addition, the following variables can be found in the variable palette and in the sensing palette a block for standarizised time: Other variables are created depending on the use of the libraries when configuring the sprites.	 SciSnap!Data SciSnap!Messages SciSnap!Properties  datetime 2021-07-02T10:08:14

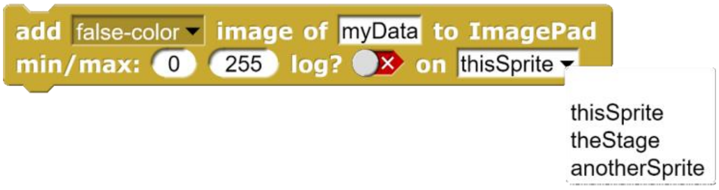
2 The structure of the SciSnap!-Sprites

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.

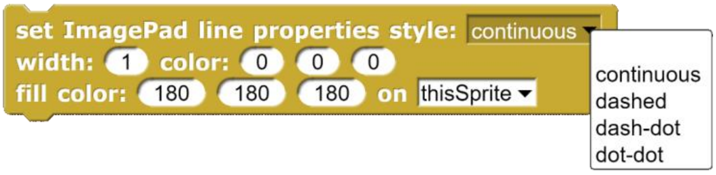
Each sprite and the stage can be configured as "special sprites", e.g. as *ImagePad*. The local variables *myProperties* and *myData* are created and some useful presets are made for the properties. The properties are usually grouped into groups, e.g. costume properties (*costumeProperties*) or the way 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.



configure thisSprite ▾ ~~as an Ima~~gePad width: 400
height: 300 colo… 5 245



add false-color ▾ image of myData to ImagePad
min/max: 0 255 log? ☒ on thisSprite ▾



set ImagePad line properties style: continuous ▾
width: 1 color: 0 0 0
fill color: 180 180 180 on thisSprite ▾

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.

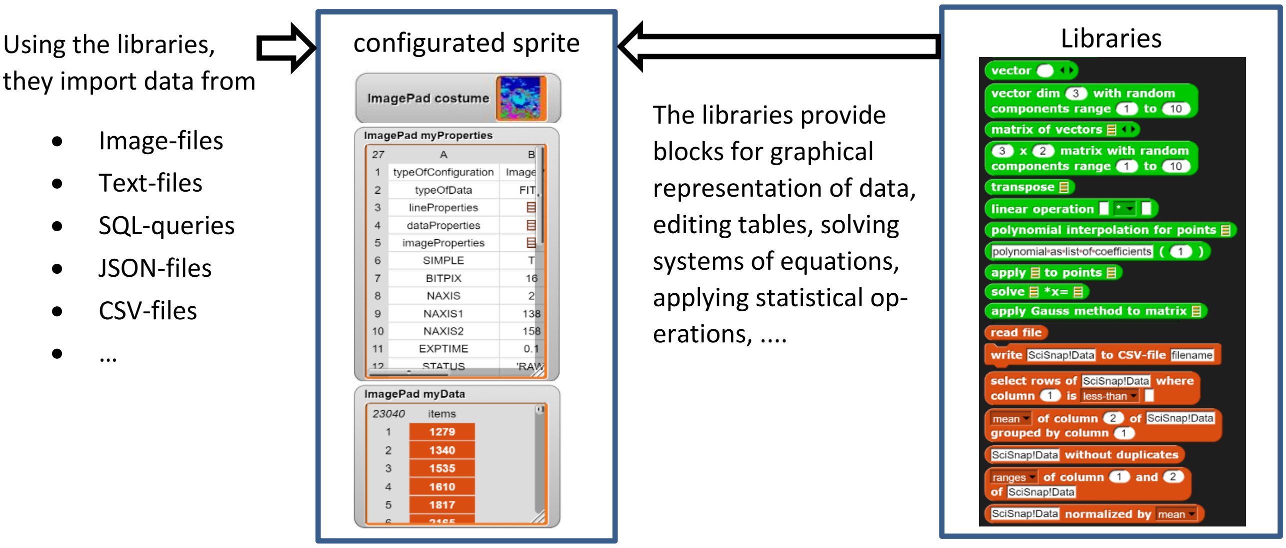
We adapt this structure by assigning two local variables to a *SciSnap!* sprite, each containing the data (*myData*) and the data description (*myProperties*). These variables can be filled by importing data from different sources (SQL query, text file, CVS file, JSON file, FITS file, direct assignment, ...), 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
- …

configurated sprite

ImagePad costume

ImagePad myProperties

27	A	B
1	typeOfConfiguration	Image
2	typeOfData	FIT
3	lineProperties	☰
4	dataProperties	☰
5	imageProperties	☰
6	SIMPLE	T
7	BITPIX	16
8	NAXIS	2
9	NAXIS1	138
10	NAXIS2	158
11	EXPTIME	0.1
12	STATUS	'RAW

ImagePad myData

23040	items
1	1279
2	1340
3	1535
4	1610
5	1817
6	2165

The libraries provide blocks for graphical representation of data, editing tables, solving systems of equations, applying statistical operations, ...

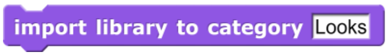
Libraries

```
vector
vector dim (3) with random components range (1) to (10)
matrix of vectors ☰
(3) x (2) matrix with random components range (1) to (10)
transpose ☰
linear operation [] [* ▾] []
polynomial interpolation for points ☰
[polynomial as list of coefficients] ( (1) )
apply ☰ to points ☰
solve ☰ *x= ☰
apply Gauss method to matrix ☰
read file
write [SciSnap!Data] to CSV-file [filename]
select rows of [SciSnap!Data] where column (1) is [less-than ▾] []
[mean ▾] of column (2) of [SciSnap!Data] grouped by column (1)
[SciSnap!Data] without duplicates
[ranges ▾] of column (1) and (2) of [SciSnap!Data]
[SciSnap!Data] normalized by [mean ▾]
```

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

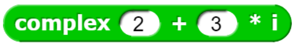
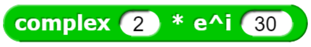
In the following, the libraries are presented in tabular form. More extensive examples, which mostly use several libraries, follow afterwards.

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 Looks can be used for this. With its help, the libraries can also be used in a "normal", i.e. unmodified *Snap!* version.

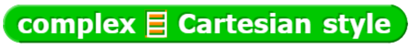
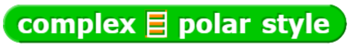
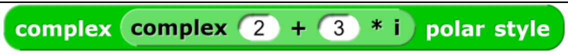
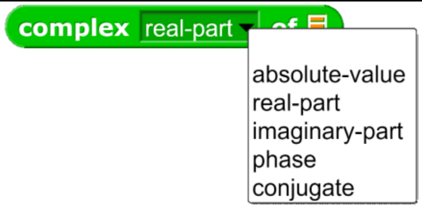
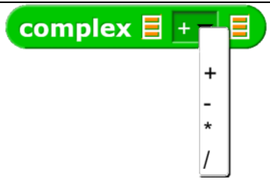
3.1 The Mathematics-Libraries (SciSnap!FullMathLibrary.xml)

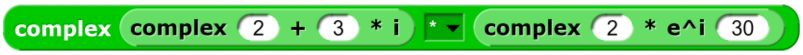
3.1.1 **Complex Numbers** (SciSnap!ComplexNumbersLibrary.xml)

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:

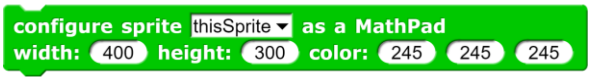
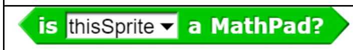
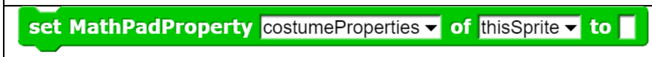
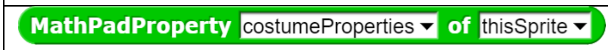
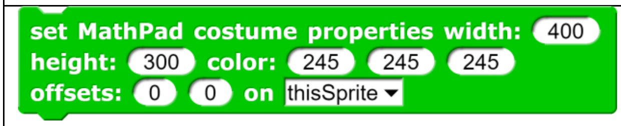
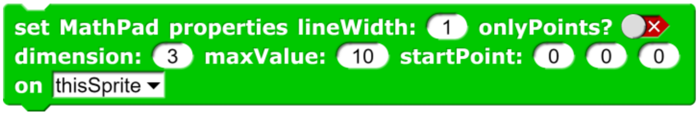
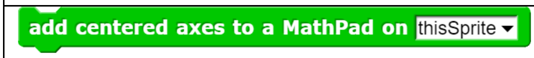
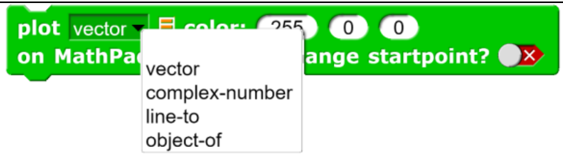
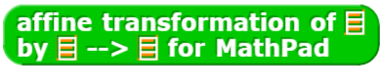
 complex 2 + 3 * i	Returns a complex number in Cartesian form.
 complex 2 * e^i 30	Returns a complex number in polar form.

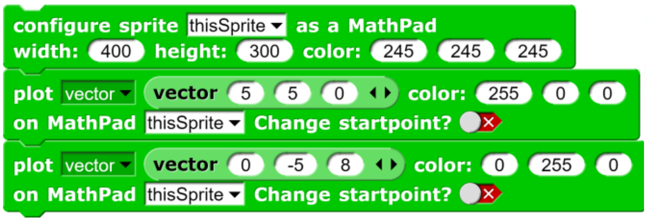
If necessary, these forms of representation can be converted into each other, and arithmetic operations can be performed.

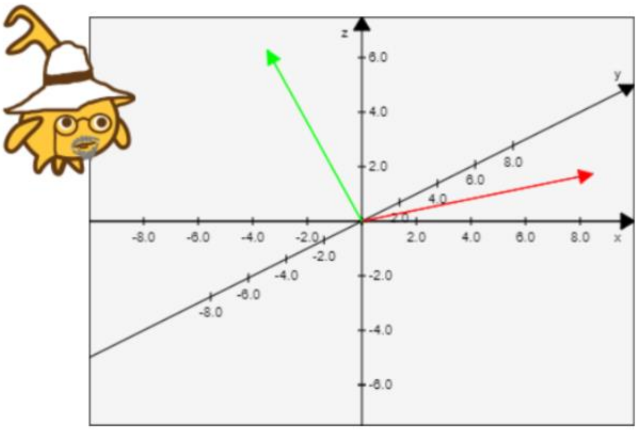
 complex ☰ Cartesian style	Returns a complex number in Cartesian form.
 complex ☰ polar style	Returns a complex number in polar form.
 complex complex 2 + 3 * i polar style	Example of format conversion.
 complex real-part of ☰ absolute-value real-part imaginary-part phase conjugate	The components of a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

Example of multiplication:  complex complex 2 + 3 * i * complex 2 * e^i 30

3.1.2 The MathPad (SciSnap!MathPadLibrary.xml)

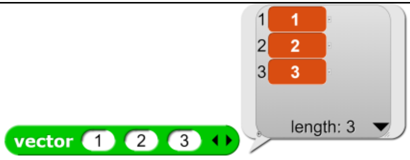
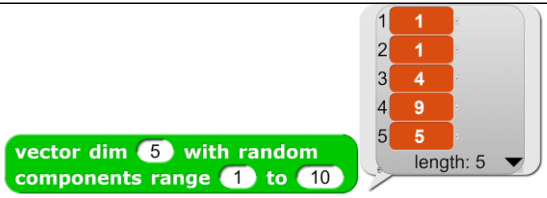
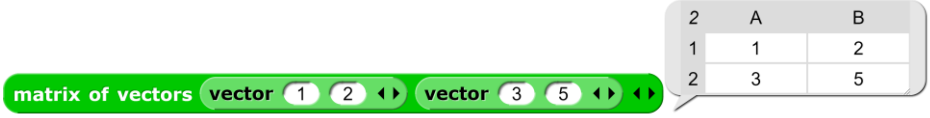
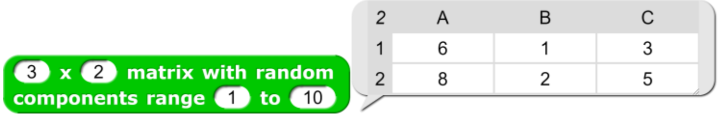
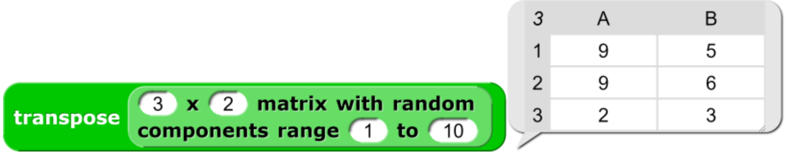
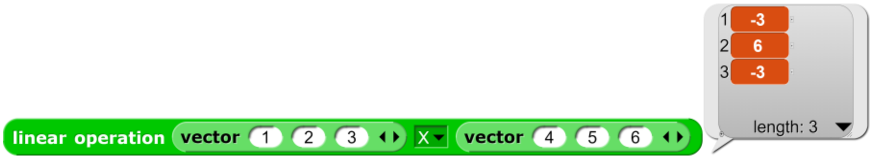
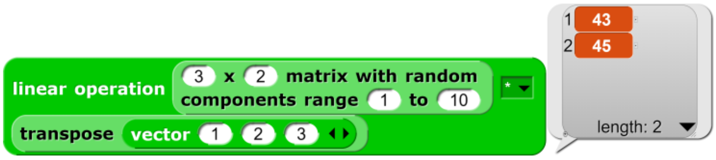
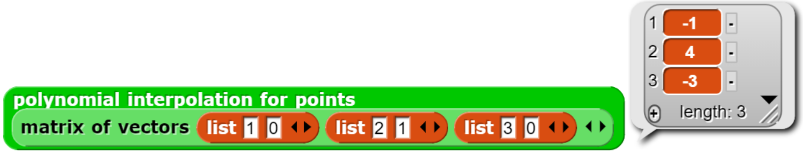
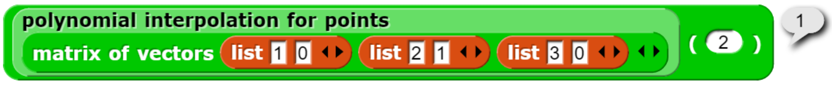
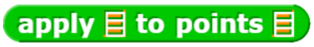
 configure sprite thisSprite ▾ as a MathPad width: 400 height: 300 color: 245 245 245	Configures a sprite as a MathPad and draws a 3-dimensional coordinate system centered in the middle.
 is thisSprite ▾ a MathPad?	Test for MathPad configuration.
 set MathPadProperty costumeProperties ▾ of thisSprite ▾ to	Sets a MathPad property.
 MathPadProperty costumeProperties ▾ of thisSprite ▾	Reads a MathPad property.
 set MathPad costume properties width: 400 height: 300 color: 245 245 245 offsets: 0 0 on thisSprite ▾	Sets the costume properties of a MathPad to the specified values.
 set MathPad properties lineWidth: 1 onlyPoints? ☒ dimension: 3 maxValue: 10 startPoint: 0 0 0 on thisSprite ▾	Sets the line properties of a MathPad to the specified values. If "onlyPoints" is set, only the endpoints are drawn.
 add centered axes to a MathPad on thisSprite ▾	Draws the coordinate system on a MathPad.
 plot vector ▾ ☰ color: 255 0 0 on MathPa ange startpoint? ☒ vector complex-number line-to object-of	Draws a vector, complex number, line or object described by a list of vectors.
 affine transformation of ☰ by ☰ --> ☰ for MathPad	Performs an affine transformation in the plane for a list of points, e.g. an object, described by mapping three points to three others.

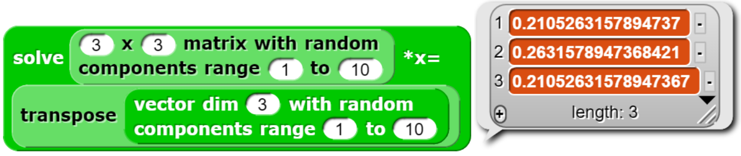
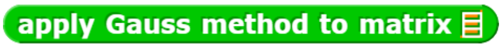
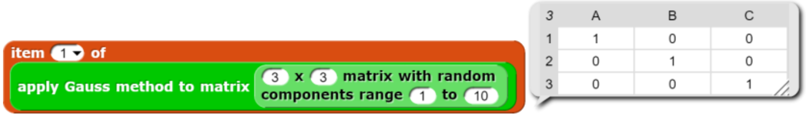

configure sprite thisSprite ▾ as a MathPad width: 400 height: 300 color: 245 245 245
plot vector ▾ vector 5 5 0 ◂▸ color: 255 0 0 on MathPad thisSprite ▾ Change startpoint? ☒
plot vector ▾ vector 0 -5 8 ◂▸ color: 0 255 0 on MathPad thisSprite ▾ Change startpoint? ☒



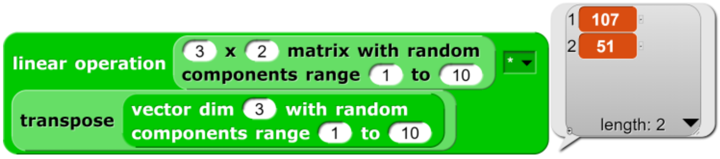
3.1.3 Linear Algebra (SciSnap!LinearAlgebraLibrary.xml)

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 1 2 3 1 1 2 2 3 3 length: 3	Returns a vector of any dimension with predefined values.
 vector dim 5 with random components range 1 to 10 1 1 2 1 3 4 4 9 5 5 length: 5	Returns a vector of arbitrary dimension with random values from the specified range.
 matrix of vectors vector 1 2 vector 3 5 2 A B 1 1 2 2 3 5	Returns the matrix from the specified vectors.
 3 x 2 matrix with random components range 1 to 10 2 A B C 1 6 1 3 2 8 2 5	Returns a matrix of arbitrary dimension with random values from the specified range.
 transpose 3 x 2 matrix with random components range 1 to 10 3 A B 1 9 5 2 9 6 3 2 3	Matrices and vectors can be transposed.
 linear operation vector 1 2 3 X vector 4 5 6 1 -3 2 6 3 -3 length: 3  linear operation 3 x 2 matrix with random components range 1 to 10 * transpose vector 1 2 3 1 43 2 45 length: 2	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 matrix of vectors list 1 0 list 2 1 list 3 0 1 -1 2 4 3 -3 length: 3	Calculates the coefficients of the polynomial through *n* points.
 polynomial interpolation for points matrix of vectors list 1 0 list 2 1 list 3 0 (2) 1	And for such polynomials one can calculate function values.
 apply ▤ to points ▤	Applies a matrix to a list of points (look at the example).

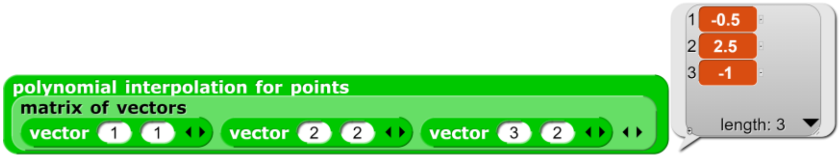
 solve 3 x 3 matrix with random components range 1 to 10 *x= transpose vector dim 3 with random components range 1 to 10 1 0.2105263157894737 2 0.2631578947368421 3 0.21052631578947367 length: 3	Calculates the solution to a system of linear equations.
 apply Gauss method to matrix	The block returns several data about the passed matrix: the diagonalized matrix, if any, its rank, whether column swaps have occurred, and the current order of the columns.
 item 1 of apply Gauss method to matrix 3 x 3 matrix with random components range 1 to 10 3 A B C 1 1 0 0 2 0 1 0 3 0 0 1	If only the diagonalized matrix is of interest, then we consider the first element of the result.

Beispiele:



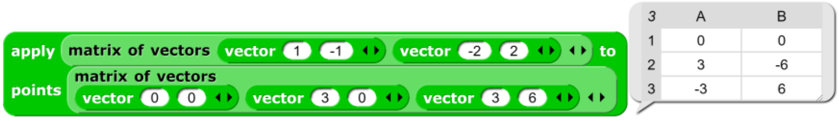
linear operation 3 x 2 matrix with random components range 1 to 10 * transpose vector dim 3 with random components range 1 to 10

1 107
2 51
length: 2



polynomial interpolation for points matrix of vectors vector 1 1 vector 2 2 vector 3 2

1 -0.5
2 2.5
3 -1
length: 3

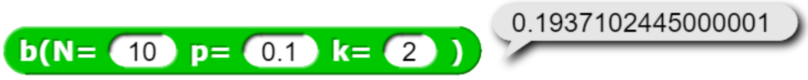
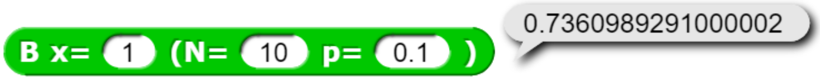
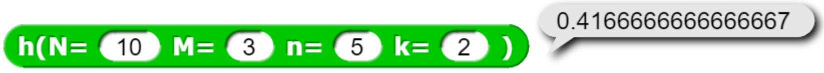
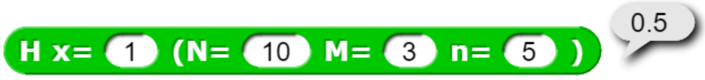
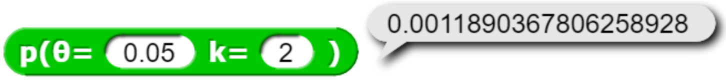
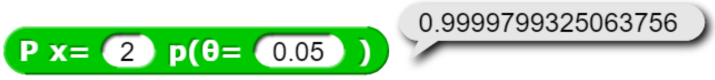
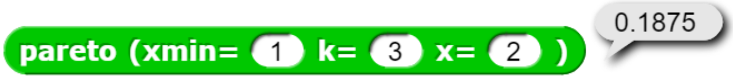
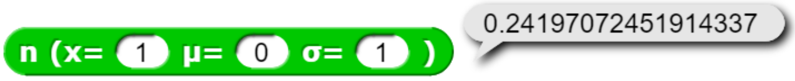


apply matrix of vectors vector 1 -1 vector -2 2 to points matrix of vectors vector 0 0 vector 3 0 vector 3 6

3	A	B
1	0	0
2	3	-6
3	-3	6

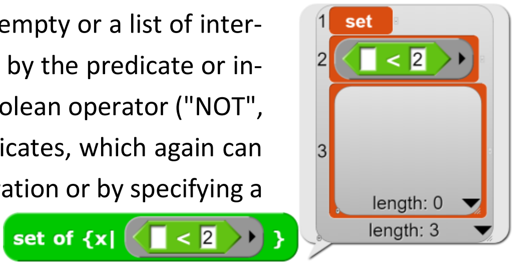
3.1.3 **Statistics** (SciSnap!StatisticsLibrary.xml)

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.

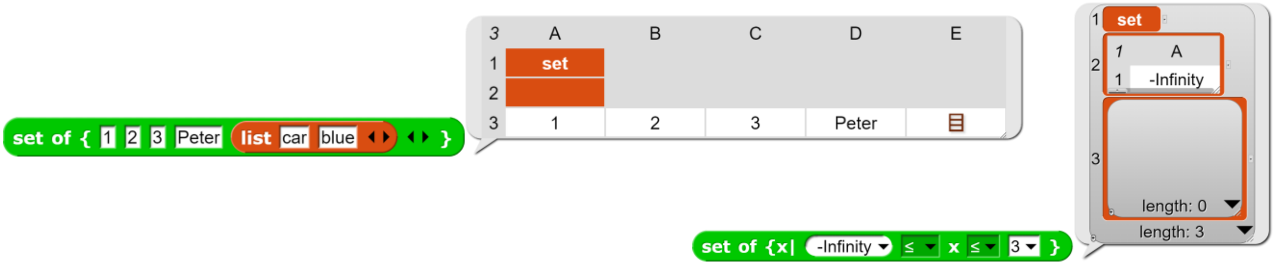
 b(N= 10 p= 0.1 k= 2) 0.1937102445000001	Probabilities of the binomial distribution $b(N,p,k)=\binom{N}{k}\cdot p^k\cdot(1-p)^{N-k}$
 B x= 1 (N= 10 p= 0.1) 0.7360989291000002	Calculates the cumulative distribution function of the binomial distribution.
 h(N= 10 M= 3 n= 5 k= 2) 0.41666666666666667	Probabilities of the hypergeometric distribution $h(N,M,n,k)=\frac{\binom{M}{k}\cdot\binom{N-M}{n-k}}{\binom{N}{n}}$
 H x= 1 (N= 10 M= 3 n= 5) 0.5	Calculates the cumulative distribution function of the hypergeometric distribution.
 p(θ= 0.05 k= 2) 0.0011890367806258928	Probabilities of the Poisson distribution $p(\theta,k)=\frac{\theta^k\cdot e^{-\theta}}{k!}$
 P x= 2 p(θ= 0.05) 0.9999799325063756	Calculates the cumulative distribution function of the Poisson distribution.
 pareto (xmin= 1 k= 3 x= 2) 0.1875	Probabilities of the Pareto distribution $pareto(x_{min},k,x)=\frac{k\cdot x_{min}{}^{k}}{x^{k+1}};$ $x\geq x_{min}; 0\ sonst$
 n (x= 1 μ= 0 σ= 1) 0.24197072451914337	Probabilities of the normal distribution $n(x,\mu,\sigma)=\frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}};$

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.


set of {x| □ < 2 }

1 set
2 □ < 2
3 length: 0
length: 3

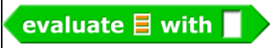
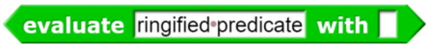
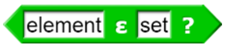
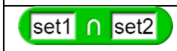
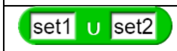
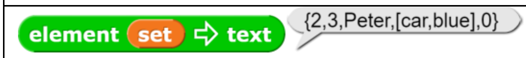
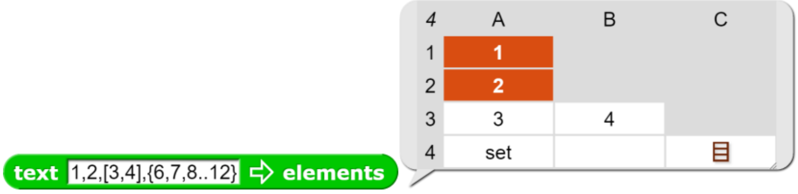

set of { 1 2 3 Peter list car blue }

3	A	B	C	D	E
1	set				
2					
3	1	2	3	Peter	▤

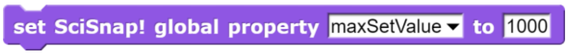
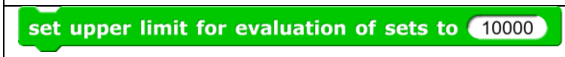
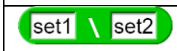
set of {x| -Infinity ≤ x ≤ 3 }

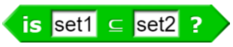
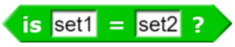
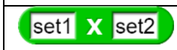
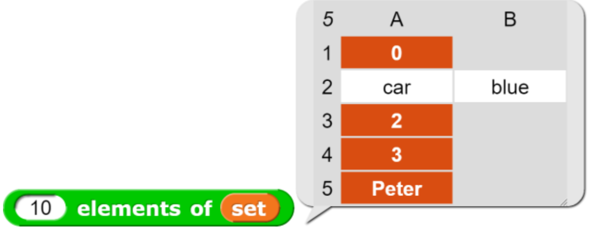
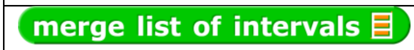
1 set
2 1 A; 1 -Infinity
3 length: 0
length: 3

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*".

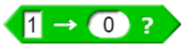
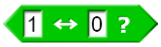
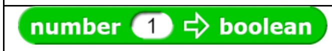
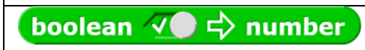
 evaluate ▤ with □  evaluate ringified predicate with □	Returns the result of the predicate applied to the passed element or an interval list.
 element ε set ?	Returns "true" if the element is an element of the set, otherwise "false".
 set1 ∩ set2	Returns the intersection of two sets.
 set1 ∪ set2	Returns the union of two sets.
 element set ⇨ text {2,3,Peter,[car,blue],0}	Represents the elements of a set "a bit more readable".
 text 1,2,[3,4],{6,7,8..12} ⇨ elements 4 A B C 1 1 2 2 3 3 4 4 set ▤	Generates the corresponding list of elements from a text. Lists are bracketed "square" in the text, sets are bracketed "curly".

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.

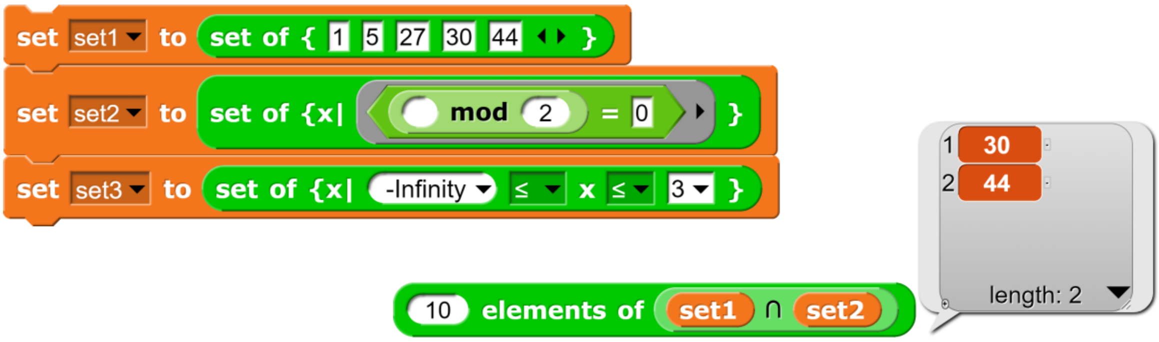
 set SciSnap! global property maxSetValue to 1000	Sets the limit up to which predicates are checked, if applicable.
 set upper limit for evaluation of sets to 10000	dito
 set1 \ set2	Returns the difference of two sets.

 is set1 ⊆ set2 ?	Returns "true" if the first set is subset of the second, otherwise "false".
 is set1 = set2 ?	Returns "true" if the sets contain the same elements, otherwise "false".
 set1 X set2	Returns the Cartesian product of two sets.
 10 elements of set 5 A B 1 0 2 car blue 3 2 4 3 5 Peter	Returns the first n elements of a set as a list.
 merge list of intervals	Merges a list of intervals.

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.

 1 → 0 ?	Implication, returns "true" if the conclusion is formally true, otherwise "false".
 1 ↔ 0 ?	Equivalence, returns "true" if the inputs are formally equivalent, otherwise "false".
 number 1 ⇨ boolean	Conversion of switch values into truth values.
 boolean ✓ ⇨ number	Conversion of truth values into switch values.

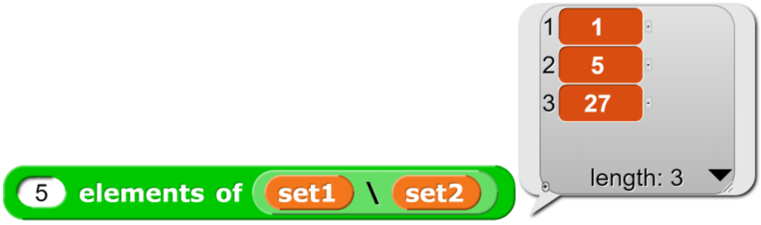
Examples:



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 }

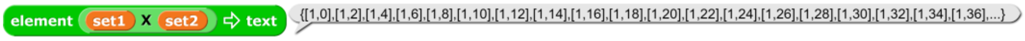
10 elements of (set1 ∩ set2)

1 30
2 44
length: 2



5 elements of (set1 \ set2)

1 1
2 5
3 27
length: 3

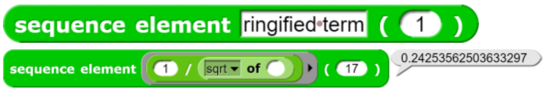
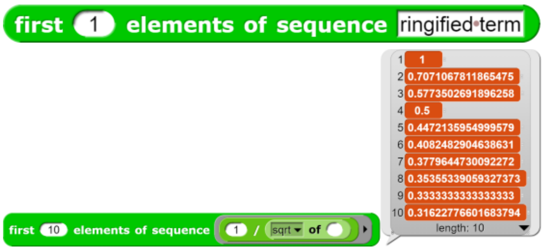
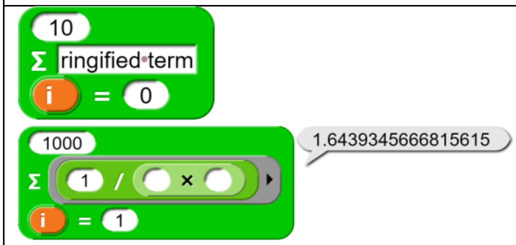
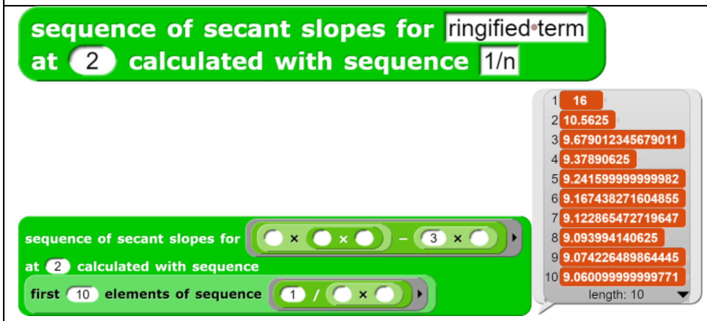
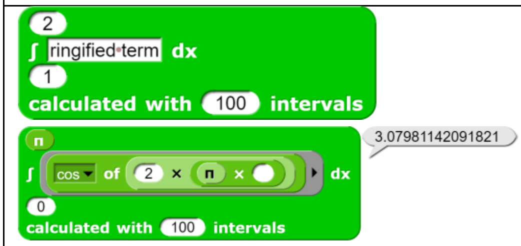
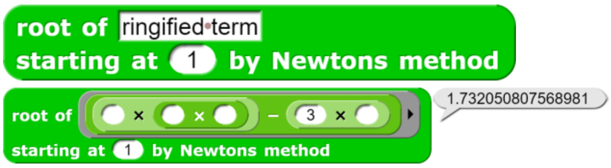
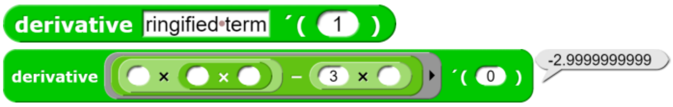


element (set1 X set2) ⇨ text

{[1,0],[1,2],[1,4],[1,6],[1,8],[1,10],[1,12],[1,14],[1,16],[1,18],[1,20],[1,22],[1,24],[1,26],[1,28],[1,30],[1,32],[1,34],[1,36],...}

3.1.5 **Numerical Methods** (SciSnap!NumericMathLibrary.xml)

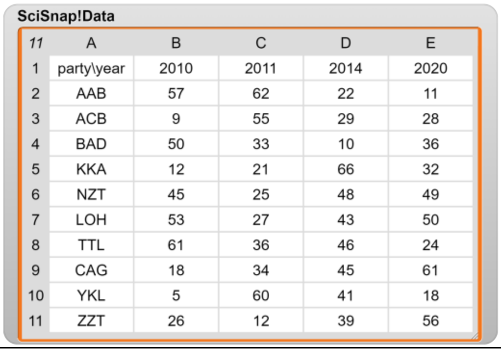
The library contains some blocks for dealing with sequences, series, secants, integrals and roots, as well as the calculation of derivatives at a given point.

 sequence element ringified term (1) sequence element (1 / sqrt of) (17) 0.24253562503633297	Calculation of a sequence element. The term must be entered with (gray) ring ("ringified"). Example: the 17th element of the sequence $\frac{1}{\sqrt{n}}$.
 first 1 elements of sequence ringified term first 10 elements of sequence (1 / sqrt of) 1 1 2 0.7071067811865475 3 0.5773502691896258 4 0.5 5 0.4472135954999579 6 0.4082482904638631 7 0.3779644730092272 8 0.35355339059327373 9 0.3333333333333333 10 0.31622776601683794 length: 10	Returns the first n members of a sequence as a list. Example: The first 10 elements of the sequence $\frac{1}{\sqrt{n}}$.
 10 Σ ringified term i = 0 1000 Σ (1 / (×)) i = 1 1.6439345666815615	Calculates the sum of a finite series. The term must be entered with (gray) ring ("ringified"). Example: The sum of the first 1000 elements of the series $\sum_{i=1}^{1000}\frac{1}{i^2}$.
 sequence of secant slopes for ringified term at 2 calculated with sequence 1/n sequence of secant slopes for (× ×) − (3 ×) at 2 calculated with sequence first 10 elements of sequence (1 / (×)) 1 16 2 10.5625 3 9.679012345679011 4 9.37890625 5 9.241599999999982 6 9.167438271604855 7 9.122865472719647 8 9.093994140625 9 9.074226489864445 10 9.060099999999771 length: 10	Sequence of secant slopes in a point. The sequence can also be specified explicitly in the form of a list. The term must be entered with (gray) ring ("ringified"). Example: 10 secant slopes near the point with x=2 of the function $f(x) = x^3 - 3x$, calculated with the sequence $\frac{1}{n^2}$.
 2 ∫ ringified term dx 1 calculated with 100 intervals π ∫ (cos of (2 × (π ×))) dx 0 calculated with 100 intervals 3.07981142091821	Numerical calculation of an integral using the trapezoidal method. The term must be entered with (gray) ring ("ringified"). Example: Calculation of the integral F=$\int_0^{\pi}\cos 2\pi x\,dx$
 root of ringified term starting at 1 by Newtons method root of ((× ×) − (3 ×)) starting at 1 by Newtons method 1.732050807568981	Root calculation according to Newton's method. The term must be entered with (gray) ring ("ringified"). Example: Calculation of the root of $f(x) = x^3 - 3x$, start at x=1.
 derivative ringified term ´(1) derivative ((× ×) − (3 ×)) ´(0) -2.9999999999	Numerical calculation of the derivative in a point. The term must be entered with (gray) ring ("ringified"). Example: Calculation of the derivative of $f(x) = x^3 - 3x$ at x=0.

3.2 The Data-Library (SciSnap!DataLibrary.xml)

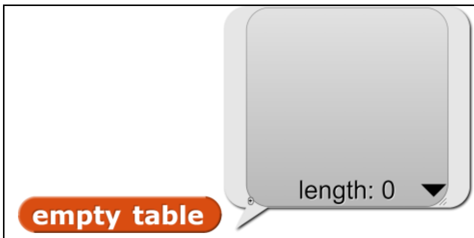
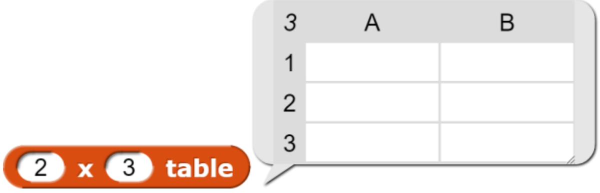
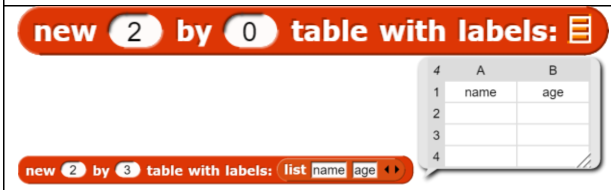
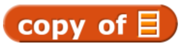
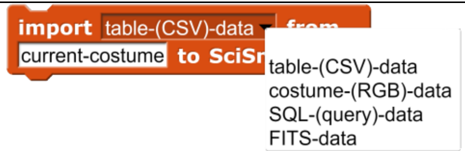
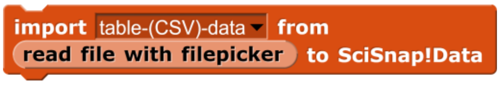
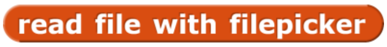
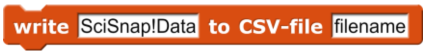
The data library of *SciSnap!* serves on the one hand for the direct manipulation also of larger data sets, on the other hand for the evaluation of data, e.g. for the computation of statistical values such as the variance or correlation. Because number structures like vectors and matrices are implemented 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.

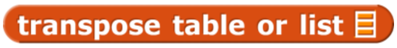
The following examples refer to a table that contains "party names" as the first column and then "election results" in the years indicated.



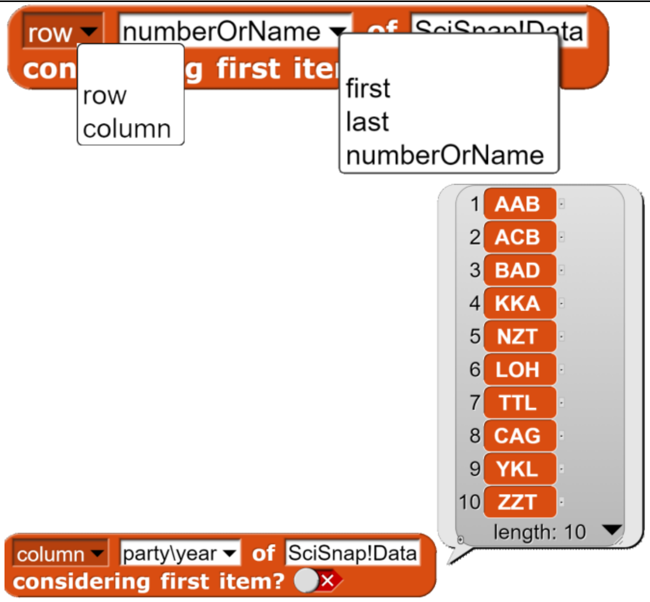
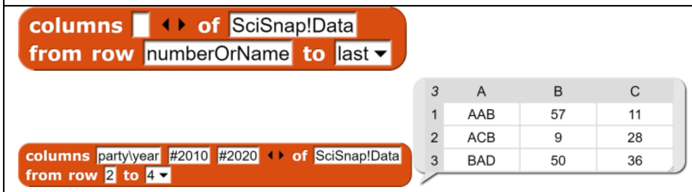
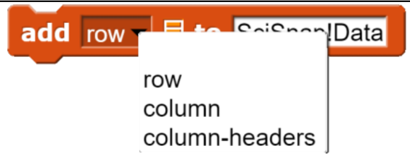
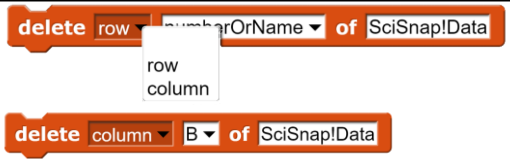
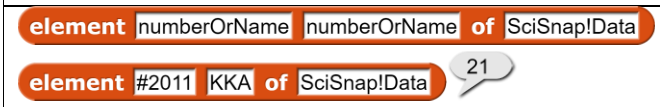
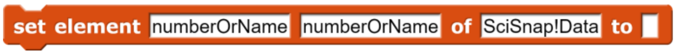
SciSnap!Data

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

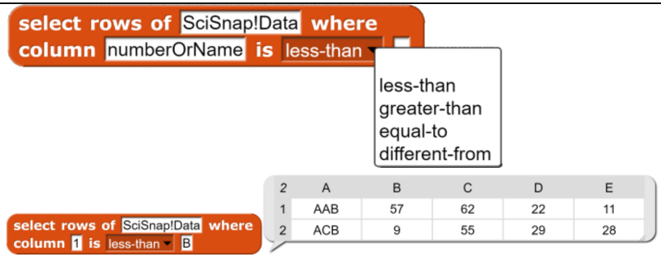
 empty table length: 0	Returns an empty table (as a start structure for further table operations).
 2 x 3 table 3 A B 1 2 3	Returns a table of the specified size without contents.
 new 2 by 0 table with labels: new 2 by 3 table with labels: list name age 4 A B 1 name age 2 3 4	Returns a table of the specified size without contents, but with column headers.
 copy of	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.
 import table-(CSV)-data from current-costume to SciSr table-(CSV)-data costume-(RGB)-data SQL-(query)-data FITS-data  import table-(CSV)-data from read file with filepicker to SciSnap!Data	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.
 read file with filepicker	Starts the file selection dialog.
 write SciSnap!Data to CSV-file filename	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.

 row ▾ numberOrName ▾ of SciSnap!Data considering first item? row column first last numberOrName column ▾ party\year ▾ of SciSnap!Data considering first item? ✕ 1 AAB 2 ACB 3 BAD 4 KKA 5 NZT 6 LOH 7 TTL 8 CAG 9 YKL 10 ZZT length: 10	Returns a row or column of the specified table. Example: The first column of the example table without the heading.
 columns ◂▸ of SciSnap!Data from row numberOrName to last ▾ columns party\year #2010 #2020 ◂▸ of SciSnap!Data from row 2 to 4 ▾ 3 A B C 1 AAB 57 11 2 ACB 9 28 3 BAD 50 36	Returns the specified row range of the specified columns. Example: Results of the three specified parties from 2010 and 2020.
 add row ▾ ☰ to SciSnap!Data row column column-headers	Adds a row, a column or column headers to the specified table. Missing elements are added with empty content, "overhanging" are ignored.
 delete row ▾ numberOrName ▾ of SciSnap!Data row column delete column ▾ B ▾ of SciSnap!Data	Deletes a row or column from the specified table. Example: Deletes the column for the year 2010 from the sample table.
 element numberOrName numberOrName of SciSnap!Data element #2011 KKA of SciSnap!Data 21	Returns the specified table element. Example: Result of the party KKA in 2011.
 set element numberOrName numberOrName of SciSnap!Data to	Sets the value in a table at the specified position.

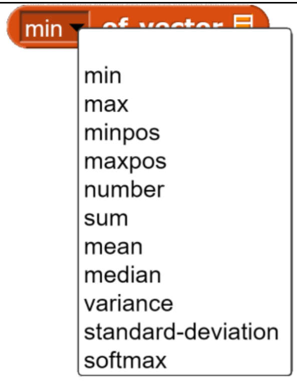
The following blocks are much more powerful. The examples refer again to the example table.

<table>
<tr>
<td>


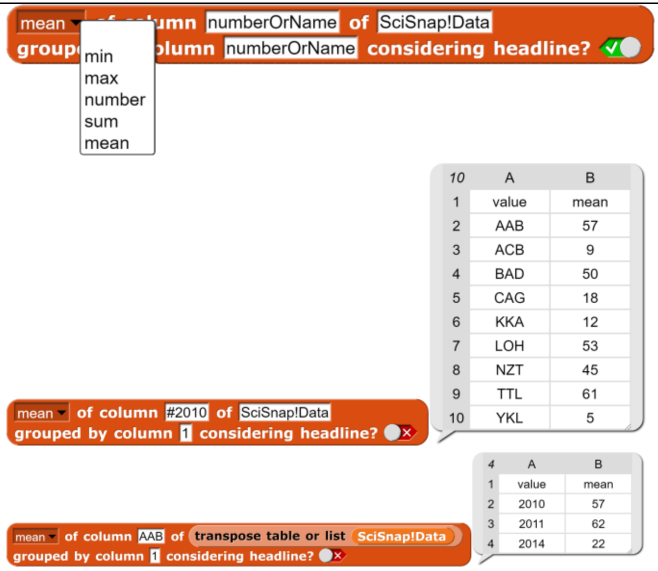
select rows of SciSnap!Data where column numberOrName is less-than ▾

select rows of SciSnap!Data where column 1 is less-than ▾ B

<table>
<tr><th>2</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th></tr>
<tr><td>1</td><td>AAB</td><td>57</td><td>62</td><td>22</td><td>11</td></tr>
<tr><td>2</td><td>ACB</td><td>9</td><td>55</td><td>29</td><td>28</td></tr>
</table>
</td>
<td>
Returns selected rows of a table that satisfy the specified criterion.

<u>Example</u>: All election results with parties beginning with "A".
</td>
</tr>
<tr>
<td>


min ▾ of vector ☰

</td>
<td>
Returns the selected property of a vector, i.e. a row or column consisting only of numbers.
</td>
</tr>
<tr>
<td>


mean ▾ of column numberOrName of SciSnap!Data grouped by column numberOrName considering headline? ✓

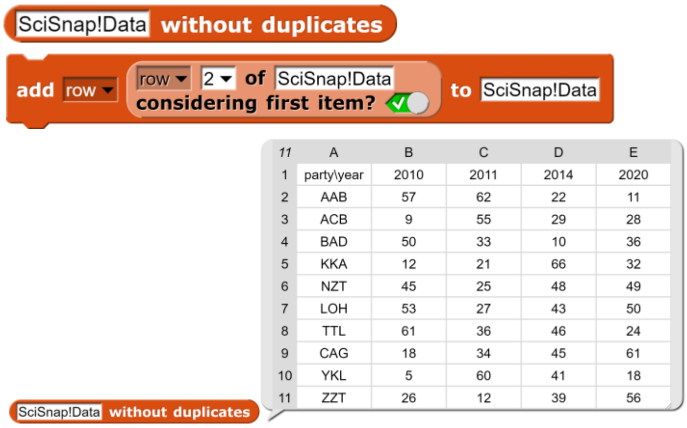
mean ▾ of column #2010 of SciSnap!Data grouped by column 1 considering headline? ✗

<table>
<tr><th>10</th><th>A</th><th>B</th></tr>
<tr><td>1</td><td>value</td><td>mean</td></tr>
<tr><td>2</td><td>AAB</td><td>57</td></tr>
<tr><td>3</td><td>ACB</td><td>9</td></tr>
<tr><td>4</td><td>BAD</td><td>50</td></tr>
<tr><td>5</td><td>CAG</td><td>18</td></tr>
<tr><td>6</td><td>KKA</td><td>12</td></tr>
<tr><td>7</td><td>LOH</td><td>53</td></tr>
<tr><td>8</td><td>NZT</td><td>45</td></tr>
<tr><td>9</td><td>TTL</td><td>61</td></tr>
<tr><td>10</td><td>YKL</td><td>5</td></tr>
</table>

mean ▾ of column AAB of transpose table or list SciSnap!Data grouped by column 1 considering headline? ✗

<table>
<tr><th>4</th><th>A</th><th>B</th></tr>
<tr><td>1</td><td>value</td><td>mean</td></tr>
<tr><td>2</td><td>2010</td><td>57</td></tr>
<tr><td>3</td><td>2011</td><td>62</td></tr>
<tr><td>4</td><td>2014</td><td>22</td></tr>
</table>
</td>
<td>
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.

<u>Examples</u>:
The mean values for 2010, grouped by party.

The mean values of the party AAB, grouped by years.
</td>
</tr>
<tr>
<td>


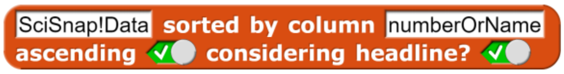
SciSnap!Data without duplicates

add row ▾ row ▾ 2 ▾ of SciSnap!Data considering first item? ✓ to SciSnap!Data

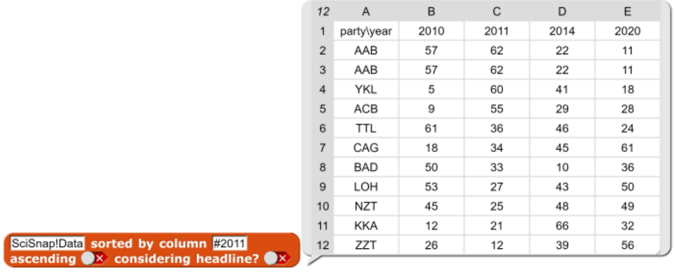
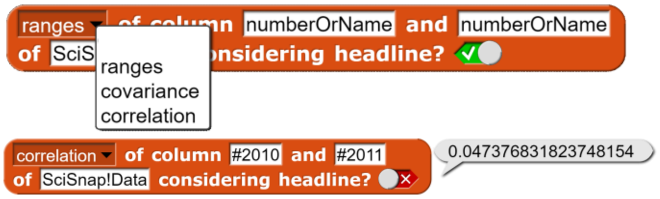
<table>
<tr><th>11</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th></tr>
<tr><td>1</td><td>party\year</td><td>2010</td><td>2011</td><td>2014</td><td>2020</td></tr>
<tr><td>2</td><td>AAB</td><td>57</td><td>62</td><td>22</td><td>11</td></tr>
<tr><td>3</td><td>ACB</td><td>9</td><td>55</td><td>29</td>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

SciSnap!Data without duplicates
</td>
<td>
Returns a table without duplicate rows or a vector without duplicates.

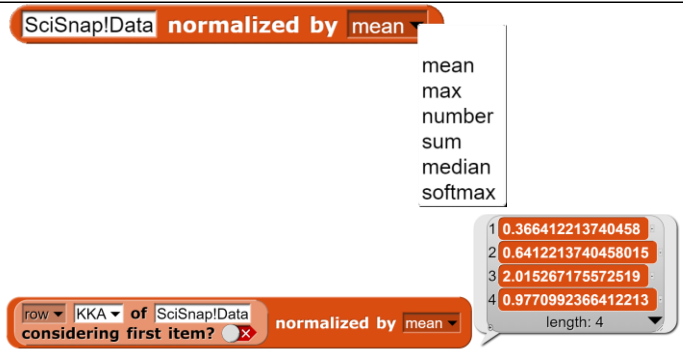
<u>Example</u>: We insert the first row of the table again at the end ...

...and then delete the duplicates again.
</td>
</tr>
<tr>
<td>


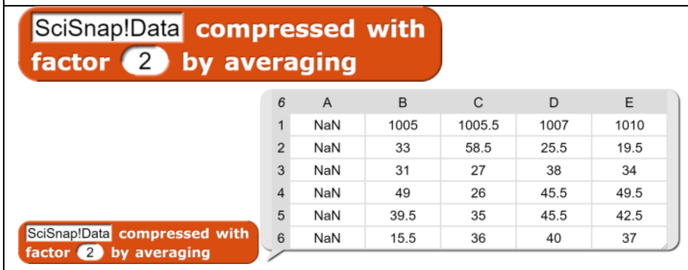
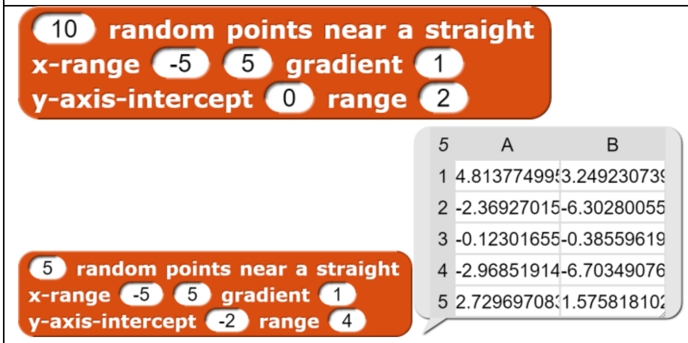
SciSnap!Data sorted by column numberOrName ascending ✓ considering headline? ✓
</td>
<td>
Returns a table sorted by the specified column in ascending or descending order.
</td>
</tr>
</table>

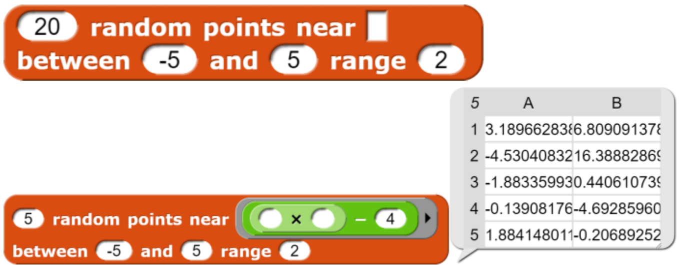
<table>
<tr>
<td>

SciSnap!Data sorted by column #2011 ascending ○✕ considering headline? ○✕
<table>
<tr><th>12</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th></tr>
<tr><td>1</td><td>party\year</td><td>2010</td><td>2011</td><td>2014</td><td>2020</td></tr>
<tr><td>2</td><td>AAB</td><td>57</td><td>62</td><td>22</td><td>11</td></tr>
<tr><td>3</td><td>AAB</td><td>57</td><td>62</td><td>22</td><td>11</td></tr>
<tr><td>4</td><td>YKL</td><td>5</td><td>60</td><td>41</td><td>18</td></tr>
<tr><td>5</td><td>ACB</td><td>9</td><td>55</td><td>29</td><td>28</td></tr>
<tr><td>6</td><td>TTL</td><td>61</td><td>36</td><td>46</td><td>24</td></tr>
<tr><td>7</td><td>CAG</td><td>18</td><td>34</td><td>45</td><td>61</td></tr>
<tr><td>8</td><td>BAD</td><td>50</td><td>33</td><td>10</td><td>36</td></tr>
<tr><td>9</td><td>LOH</td><td>53</td><td>27</td><td>43</td><td>50</td></tr>
<tr><td>10</td><td>NZT</td><td>45</td><td>25</td><td>48</td><td>49</td></tr>
<tr><td>11</td><td>KKA</td><td>12</td><td>21</td><td>66</td><td>32</td></tr>
<tr><td>12</td><td>ZZT</td><td>26</td><td>12</td><td>39</td><td>56</td></tr>
</table>
</td>
<td>Example: The election results sorted by the year 2011.</td>
</tr>
<tr>
<td>

ranges ▾ of column numberOrName and numberOrName of SciS nsidering headline? ✓○

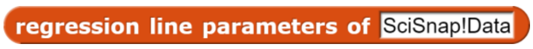
correlation ▾ of column #2010 and #2011 of SciSnap!Data considering headline? ○✕
0.047376831823748154
</td>
<td>Returns ranges, covariance and correlation for two table columns.

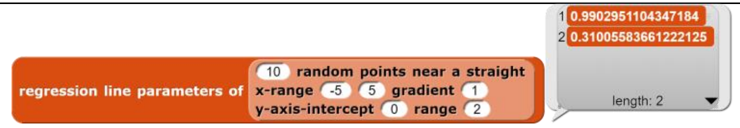
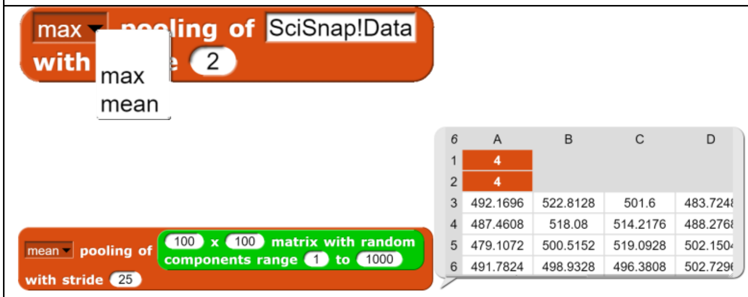
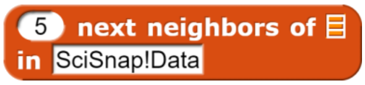
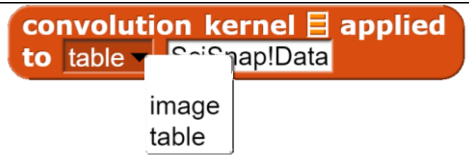
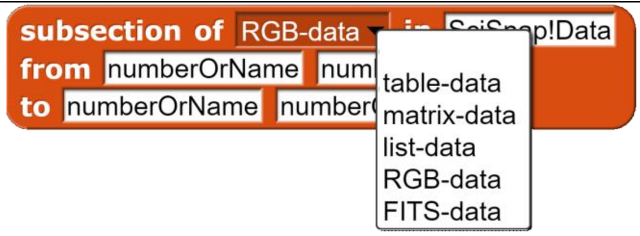
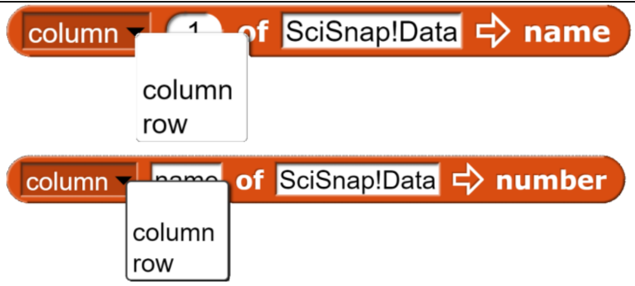
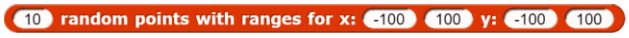
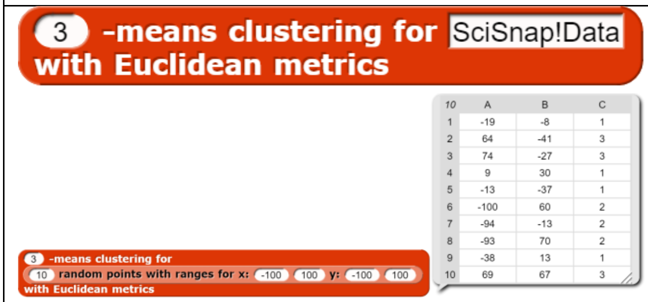
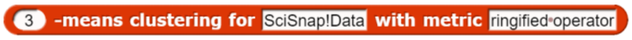
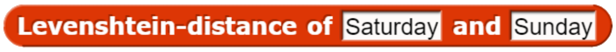
Example: Correlation for the years 2010 and 2011.</td>
</tr>
<tr>
<td>

SciSnap!Data normalized by mean ▾

row ▾ KKA ▾ of SciSnap!Data considering first item? ○✕ normalized by mean ▾
<table>
<tr><td>1</td><td>0.366412213740458</td></tr>
<tr><td>2</td><td>0.6412213740458015</td></tr>
<tr><td>3</td><td>2.015267175572519</td></tr>
<tr><td>4</td><td>0.9770992366412213</td></tr>
</table>
length: 4
</td>
<td>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.</td>
</tr>
<tr>
<td>

SciSnap!Data compressed with factor 2 by averaging
<table>
<tr><th>6</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th></tr>
<tr><td>1</td><td>NaN</td><td>1005</td><td>1005.5</td><td>1007</td><td>1010</td></tr>
<tr><td>2</td><td>NaN</td><td>33</td><td>58.5</td><td>25.5</td><td>19.5</td></tr>
<tr><td>3</td><td>NaN</td><td>31</td><td>27</td><td>38</td><td>34</td></tr>
<tr><td>4</td><td>NaN</td><td>49</td><td>26</td><td>45.5</td><td>49.5</td></tr>
<tr><td>5</td><td>NaN</td><td>39.5</td><td>35</td><td>45.5</td><td>42.5</td></tr>
<tr><td>6</td><td>NaN</td><td>15.5</td><td>36</td><td>40</td><td>37</td></tr>
</table>
SciSnap!Data compressed with factor 2 by averaging
</td>
<td>Returns a table compressed by the factor n by averaging. Texts cannot be compressed in this way, of course.</td>
</tr>
<tr>
<td>

10 random points near a straight x-range -5 5 gradient 1 y-axis-intercept 0 range 2
<table>
<tr><th>5</th><th>A</th><th>B</th></tr>
<tr><td>1</td><td>4.81377499</td><td>3.24923073</td></tr>
<tr><td>2</td><td>-2.36927015</td><td>-6.30280055</td></tr>
<tr><td>3</td><td>-0.12301655</td><td>-0.38559619</td></tr>
<tr><td>4</td><td>-2.96851914</td><td>-6.70349076</td></tr>
<tr><td>5</td><td>2.72969708</td><td>1.5758181</td></tr>
</table>
5 random points near a straight x-range -5 5 gradient 1 y-axis-intercept -2 range 4
</td>
<td>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 $f(x) = x - 2$.</td>
</tr>
<tr>
<td>

20 random points near □ between -5 and 5 range 2
<table>
<tr><th>5</th><th>A</th><th>B</th></tr>
<tr><td>1</td><td>3.18966283</td><td>6.80909137</td></tr>
<tr><td>2</td><td>-4.53040832</td><td>16.3888286</td></tr>
<tr><td>3</td><td>-1.88335993</td><td>0.44061073</td></tr>
<tr><td>4</td><td>-0.13908176</td><td>-4.69285960</td></tr>
<tr><td>5</td><td>1.88414801</td><td>-0.20689252</td></tr>
</table>
5 random points near (○ × ○ − 4) between -5 and 5 range 2
</td>
<td>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.

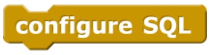
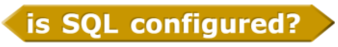
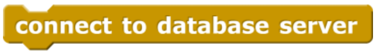
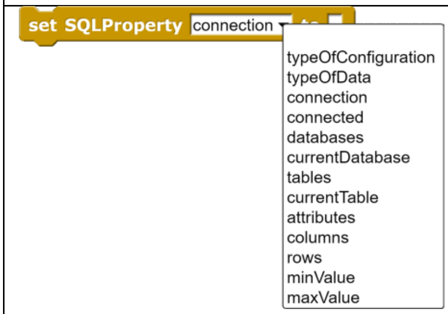
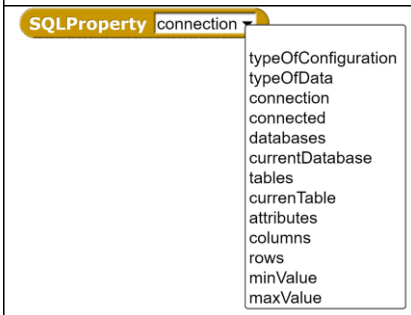
Example:
5 points scattering around $f(x) = x^2 - 4$.</td>
</tr>
<tr>
<td>

regression line parameters of SciSnap!Data
</td>
<td>Returns the slope and y-intercept of the regression line through the specified data.</td>
</tr>
</table>

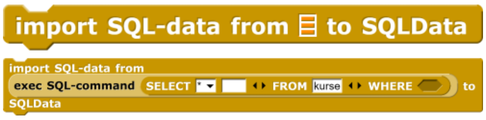
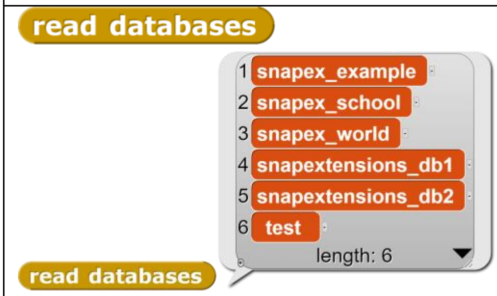
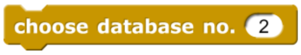
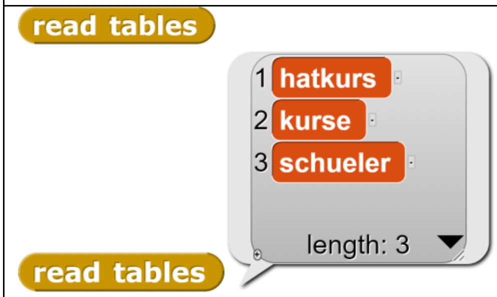
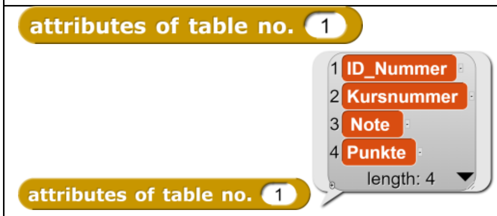
 regression line parameters of 10 random points near a straight x-range -5 5 gradient 1 y-axis-intercept 0 range 2 1 0.9902951104347184 2 0.31005583661222125 length: 2	Example: Regression line through 10 random points scattering around a straight line.
 max pooling of SciSnap!Data with 2 max mean mean pooling of 100 x 100 matrix with random components range 1 to 1000 with stride 25 6 A B C D 1 4 2 4 3 492.1696 522.8128 501.6 483.7248 4 487.4608 518.08 514.2176 488.2768 5 479.1072 500.5152 519.0928 502.1504 6 491.7824 498.9328 496.3808 502.7296	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.
 5 next neighbors of ☰ in SciSnap!Data	k-nearest neighbor (kNN) method in two dimensions for machine learning. Example: HR graph
 convolution kernel ☰ applied to table SciSnap!Data image table	Returns the result of a convolution applied either to a table or to image data. Example: Edge detection, CNN
 subsection of RGB-data in SciSnap!Data from numberOrName num to numberOrName number table-data matrix-data list-data RGB-data FITS-data	Returns a section of a table, matrix, list or image data given by the two points "left-up" and "right-down".
 column 1 of SciSnap!Data ⇨ name column row column name of SciSnap!Data ⇨ number column row	Returns the row or column number of a table with the specified label or vice versa.
 10 random points with ranges for x: -100 100 y: -100 100	Returns n random points from the specified range.
 3 -means clustering for SciSnap!Data with Euclidean metrics 3 -means clustering for 10 random points with ranges for x: -100 100 y: -100 100 with Euclidean metrics 10 A B C 1 -19 -8 1 2 64 -41 3 3 74 -27 3 4 9 30 1 5 -13 -37 1 6 -100 60 2 7 -94 -13 2 8 -93 70 2 9 -38 13 1 10 69 67 3	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.
 3 -means clustering for SciSnap!Data with metric ringified•operator	Clustering with any metric.
 Levenshtein-distance of Saturday and Sunday	Returns the Levenshtein-distance.

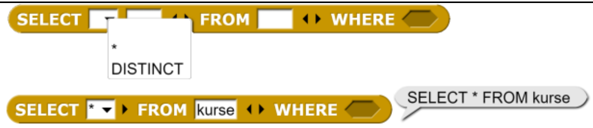
3.3 Die SQL-Library (SciSnap!SQLLibrary.xml)

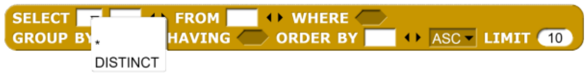
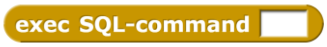
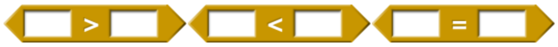
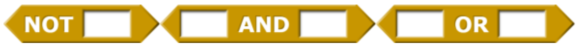
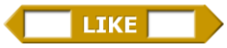
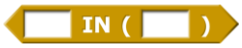
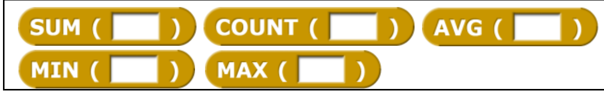
The *SQL* library contains most of the commands required for *SQL* queries (select). Other *SQL* statements can be entered directly into the *exec SQL-command* block. However, in this case you must have the appropriate access rights. The library works with two global variables *SQLData* and *SQLProperties*, 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*.

 configure SQL	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.
 is SQL configured?	Returns "true" if the configuration is correct, otherwise "false".
 connect to database server	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.
 set SQLProperty connection typeOfConfiguration typeOfData connection connected databases currentDatabase tables currentTable attributes columns rows minValue maxValue	Sets one of the properties in *SQLProperties* to the specified value.
 SQLProperty connection typeOfConfiguration typeOfData connection connected databases currentDatabase tables currenTable attributes columns rows minValue maxValue	Returns one oft the properties in *SQLProperties*.

 import SQL-data from ⌸ to SQLData import SQL-data from exec SQL-command SELECT * FROM kurse WHERE to SQLData	Imports a table into the *SQLData* data area. Example: Import of a query result.
 read databases 1 snapex_example 2 snapex_school 3 snapex_world 4 snapextensions_db1 5 snapextensions_db2 6 test length: 6	Provides a list of currently available databases.
 choose database no. 2	Selects one of the existing databases.
 read tables 1 hatkurs 2 kurse 3 schueler length: 3	Returns a list of tables in the selected database.
 attributes of table no. 1 1 ID_Nummer 2 Kursnummer 3 Note 4 Punkte length: 4	Returns a list of attributes of the specified table.

 SELECT FROM WHERE
* DISTINCT
SELECT * FROM kurse WHERE
SELECT * FROM kurse
	Block for generating simple SQL queries that can be executed by the *exec SQL-command* block.

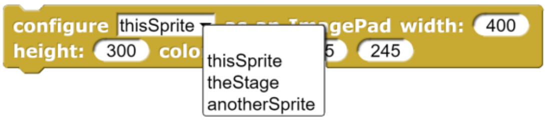
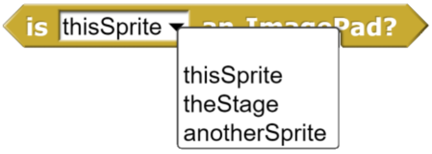
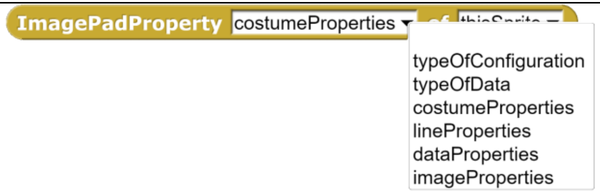
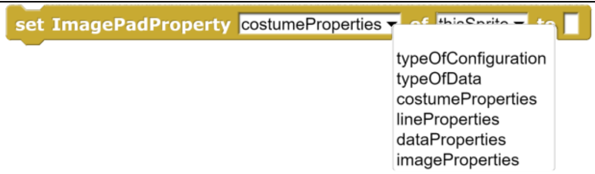
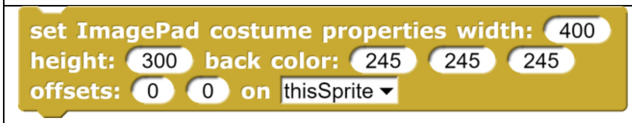
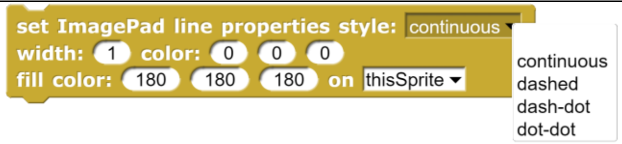
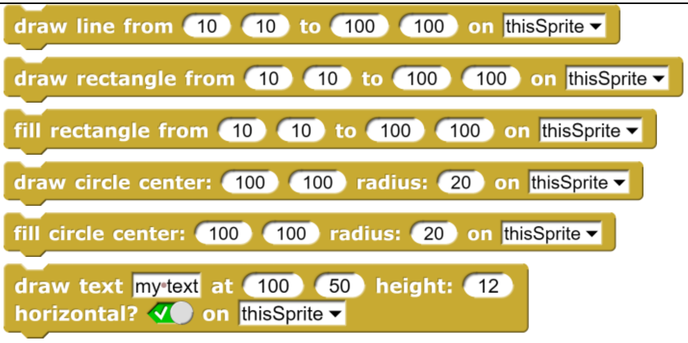
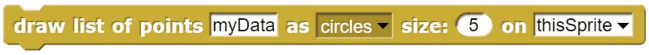
 SELECT FROM WHERE GROUP BY HAVING ORDER BY ASC LIMIT 10 * DISTINCT	Block for generating complex SQL queries that can be executed by the *exec SQL-command* block.
 exec SQL-command	Block for executing *SQL* statements to a database. The statements can be generated by *select* blocks or entered directly.
 > < =	Predicates to perform comparisons.
 NOT AND OR	Predicates to perform logical operations.
 LIKE	Predicate for checking a string pattern.
 IN ( )	Predicate to check if an element is included in items.
 SUM ( ) COUNT ( ) AVG ( ) MIN ( ) MAX ( )	Aggregate functions.

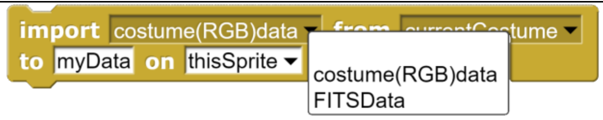
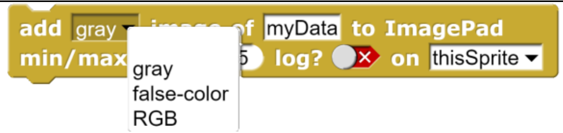
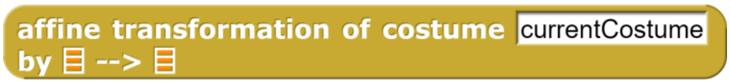
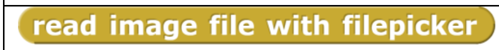
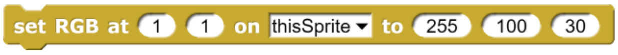
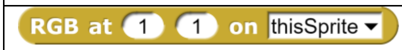
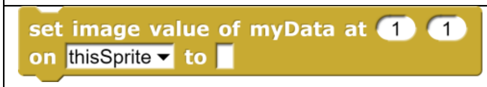
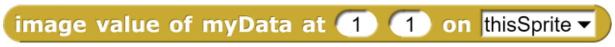
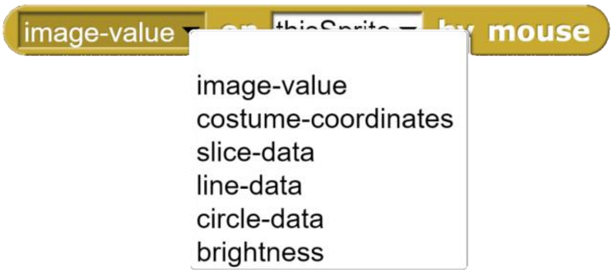
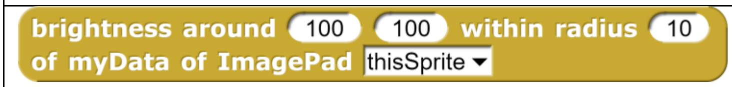
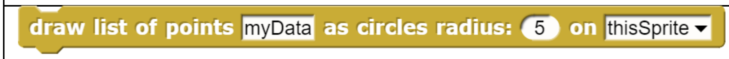
The following *SciSnap!* libraries work with locally configured *special sprites*. These are conceptually derived from *sketchpads*, so they are used for making sketches, experimenting, trying out the effect of commands, etc. If the pad is too full, then a new "page" is taken from it and work continues.

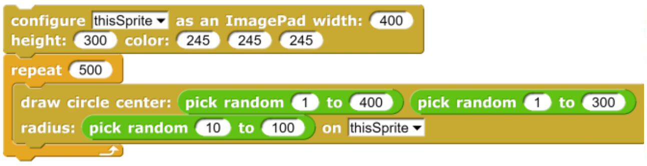
Each sprite and the stage can serve as a *special sprite*. For this purpose they are configured accordingly by creating two local variables *myData* and *myProperties* and filling them with initial values. The commands that refer to a *special sprite* all contain the target of the operation - i.e. a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

3.4 The ImagePad-Library (SciSnap!ImagePadLibrary.xml)

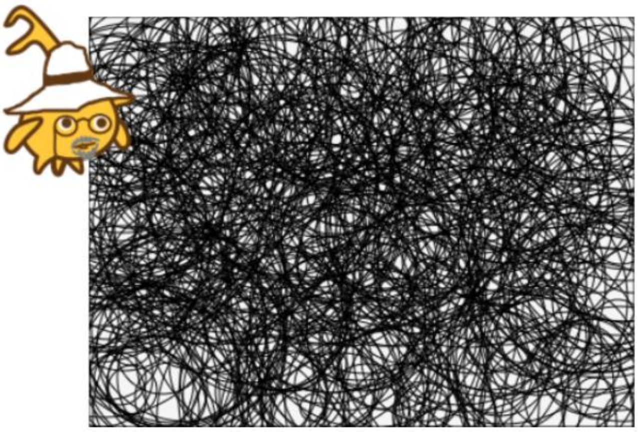
ImagePads are used to display image data, i.e. to generate images. Measurements can be made in these images using the mouse - for example, a section can be created through the image. In addition, some of the usual operations for drawing lines, rectangles, circles and texts are available. The coordinate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

 configure thisSprite ▾ gePad width: 400 height: 300 colo 5 245 thisSprite theStage anotherSprite	Configures a sprite or the stage as *ImagePad*. The name of "*anotherSprite*" must be specified if required. The command must be executed once before working with a sprite as *ImagePad*. The target of the call takes a rectangular costume with the specified dimensions and colors.
 is thisSprite ▾ ad? thisSprite theStage anotherSprite	Returns "true" if the configuration is correct, otherwise "false".
 ImagePadProperty costumeProperties ▾ typeOfConfiguration typeOfData costumeProperties lineProperties dataProperties imageProperties	Returns one of the properties in *myProperties*.
 set ImagePadProperty costumeProperties ▾ typeOfConfiguration typeOfData costumeProperties lineProperties dataProperties imageProperties	Sets one of the properties in *myProperties* to the specified value.
 set ImagePad costume properties width: 400 height: 300 back color: 245 245 245 offsets: 0 0 on thisSprite ▾	Sets the costume properties of the specified target.
 set ImagePad line properties style: continuous width: 1 color: 0 0 0 fill color: 180 180 180 on thisSprite ▾ continuous dashed dash-dot dot-dot	Sets the line properties of the specified target.
 draw line from 10 10 to 100 100 on thisSprite ▾ draw rectangle from 10 10 to 100 100 on thisSprite ▾ fill rectangle from 10 10 to 100 100 on thisSprite ▾ draw circle center: 100 100 radius: 20 on thisSprite ▾ fill circle center: 100 100 radius: 20 on thisSprite ▾ draw text my text at 100 50 height: 12 horizontal? ✓ on thisSprite ▾	elementary drawing operations
 draw list of points myData as circles ▾ size: 5 on thisSprite ▾	Draws a list of "points" as circles or squares. Attention: JS-coordinates are used!

 import costume(RGB)data from currentCostume to myData on thisSprite costume(RGB)data FITSData	Imports image data.
 add gray image of myData to ImagePad min/max 5 log? on thisSprite gray false-color RGB	Creates an image from the image data on a *ImagePad*.
 affine transformation of costume currentCostume by -->	Returns the result of an affine transformation of a costume described by specifying three original points and three image points.
 read image file with filepicker	Starts the file selection dialog.
 set RGB at 1 1 on thisSprite to 255 100 30	Sets a pixel of the costume to the specified value.
 RGB at 1 1 on thisSprite	Returns the value of a pixel of the costume.
 set image value of myData at 1 1 on thisSprite to	Sets an image point in the data area to the specified value.
 image value of myData at 1 1 on thisSprite	Returns the value of an image point from the data area.
 image-value on thisSprite by mouse image-value costume-coordinates slice-data line-data circle-data brightness	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.
 brightness around 100 100 within radius 10 of myData of ImagePad thisSprite	Returns the total brightness around a pixel in the specified radius.
 draw list of points myData as circles radius: 5 on thisSprite	Draws a list of "point" as "balls". Attention: JS-coordinates are used!

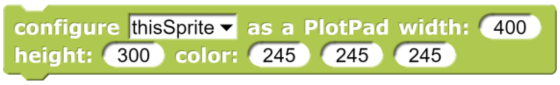
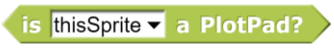
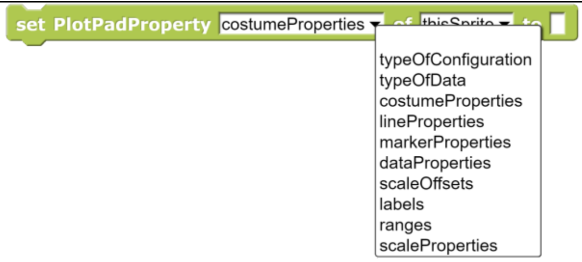
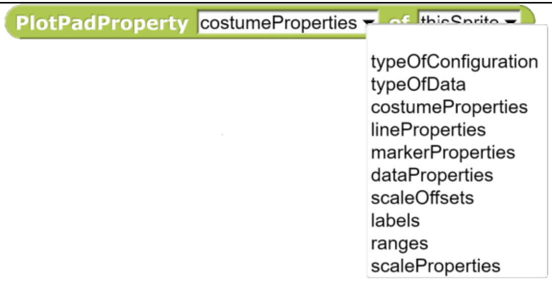
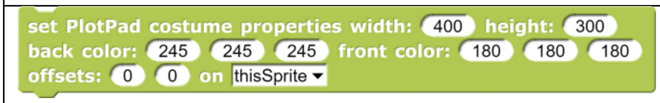
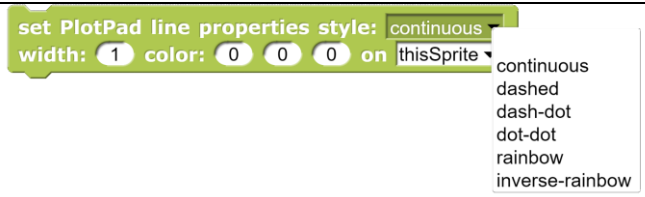
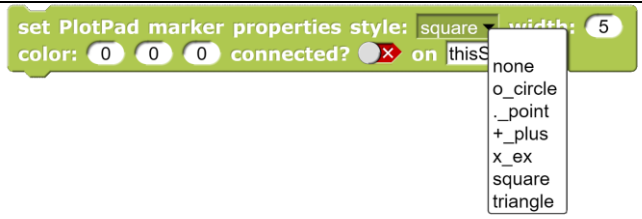
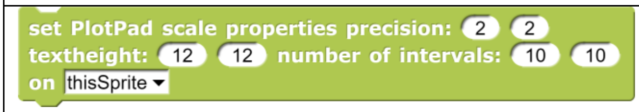
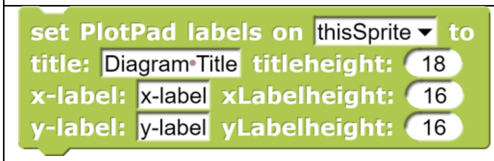


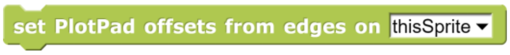
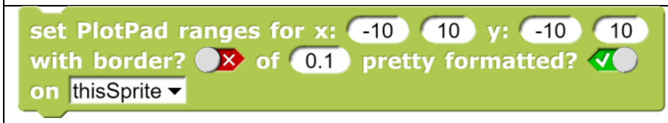
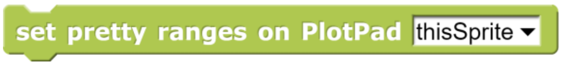
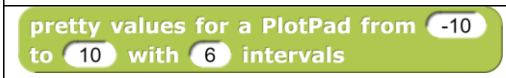
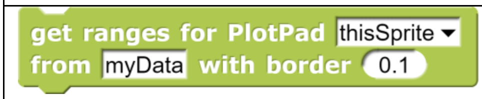
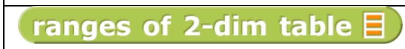
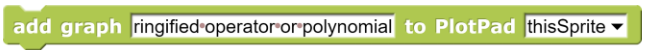
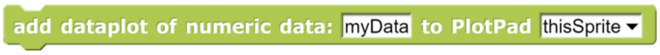
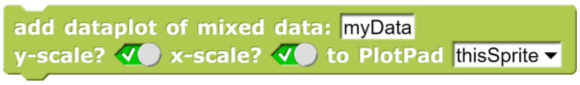
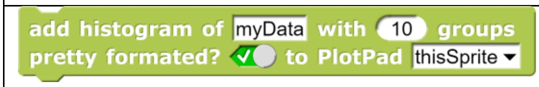
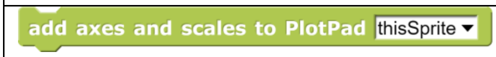
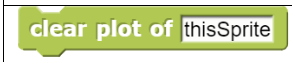
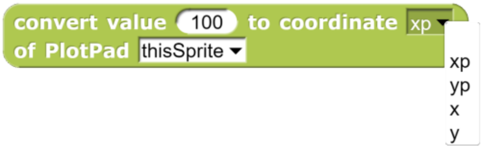
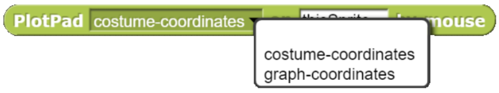
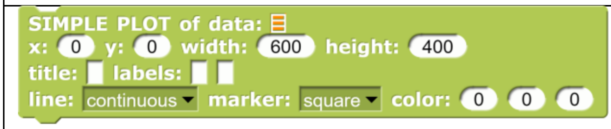
```
configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
repeat 500
  draw circle center: pick random 1 to 400 pick random 1 to 300
  radius: pick random 10 to 100 on thisSprite
```

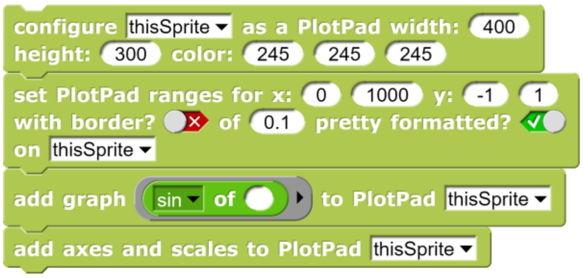


3.5 The PlotPad-Library (SciSnap!PlotPadLibrary.xml)

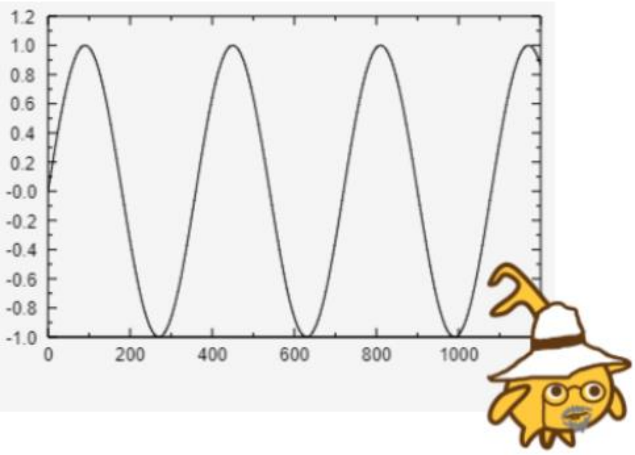
PlotPads are used to display graphs, histograms, etc. The current *Sci-Snap!PlotPadLibrary* was heavily designed by *Rick Hessman*, especially the *PrettyPrinting* and the *line styles* are from him. Many thanks for that!

 configure thisSprite as a PlotPad width: 400 height: 300 color: 245 245 245	Configures a sprite or the stage as a *PlotPad*. The name of "*anotherSprite*" 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.
 is thisSprite a PlotPad?	Returns "true" if the configuration is correct, otherwise "false".
 set PlotPadProperty costumeProperties of thisSprite to typeOfConfiguration typeOfData costumeProperties lineProperties markerProperties dataProperties scaleOffsets labels ranges scaleProperties	Sets one of the properties in *myProperties* to the specified value.
 PlotPadProperty costumeProperties of thisSprite typeOfConfiguration typeOfData costumeProperties lineProperties markerProperties dataProperties scaleOffsets labels ranges scaleProperties	Returns one of the properties in *myProperties*.
 set PlotPad costume properties width: 400 height: 300 back color: 245 245 245 front color: 180 180 180 offsets: 0 0 on thisSprite	Sets the costume properties of the specified target.
 set PlotPad line properties style: continuous width: 1 color: 0 0 0 on thisSprite continuous dashed dash-dot dot-dot rainbow inverse-rainbow	Sets the line properties of the specified target.
 set PlotPad marker properties style: square width: 5 color: 0 0 0 connected? on thisS none o_circle ._point +_plus x_ex square triangle	Sets the marker (datapoint) properties of the specified target.
 set PlotPad scale properties precision: 2 2 textheight: 12 12 number of intervals: 10 10 on thisSprite	Sets the scale properties of the specified target.
 set PlotPad labels on thisSprite to title: Diagram Title titleheight: 18 x-label: x-label xLabelheight: 16 y-label: y-label yLabelheight: 16	Sets the label properties of the specified target.

 set PlotPad offsets from edges on thisSprite	Calculates the distances of the coordinate axes from the edges.
 set PlotPad ranges for x: -10 10 y: -10 10 with border? ☒ of 0.1 pretty formatted? ✓ on thisSprite	Sets the ranges of the axes of the coordinate system.
 set pretty ranges on PlotPad thisSprite	Sets the number ranges so that "pretty" labels are on the axes.
 pretty values for a PlotPad from -10 to 10 with 6 intervals	Returns values for "pretty" labels of the axes.
 get ranges for PlotPad thisSprite from myData with border 0.1	Redetermines the ranges, calculated from the data.
 ranges of 2-dim table	Returns the ranges of a two-dimensional table.
 add graph ringified·operator·or·polynomial to PlotPad thisSprite	Adds a function graph to the *PlotPad*, given as a list of polynomial coefficients or as a "ringified" term.
 add dataplot of numeric data: myData to PlotPad thisSprite	Adds a data plot for a two-dimensional data table to the *PlotPad*.
 add dataplot of mixed data: myData y-scale? ✓ x-scale? ✓ to PlotPad thisSprite	Adds a data plot to the *PlotPad* for a two-dimensional table containing texts in the first column and numerical values in the second.
 add histogram of myData with 10 groups pretty formated? ✓ to PlotPad thisSprite	Adds a histogram to the *PlotPad*.
 add axes and scales to PlotPad thisSprite	Adds axes and labels to the *PlotPad*.
 clear plot of thisSprite	Deletes the previous plots.
 convert value 100 to coordinate xp of PlotPad thisSprite xp yp x y	Returns the conversion of a numerical value into coordinates of the *plotpad* or the coordinate system used.
 PlotPad costume-coordinates … mouse costume-coordinates graph-coordinates	Converts the mouse position into costume or graph coordinates.
 SIMPLE PLOT of data: x: 0 y: 0 width: 600 height: 400 title: labels: line: continuous marker: square color: 0 0 0	Simple plot program for data.

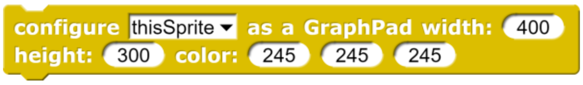
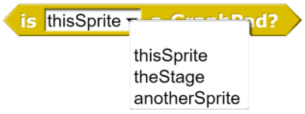
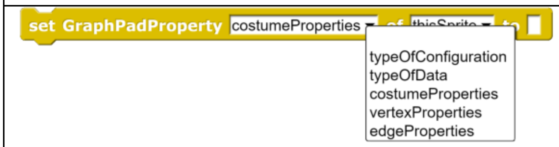
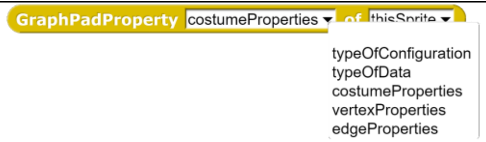
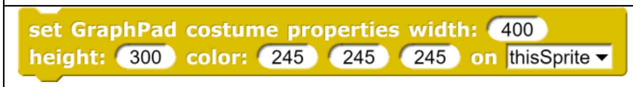
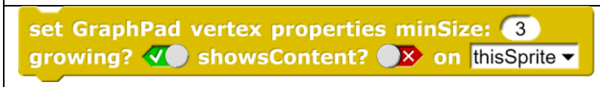
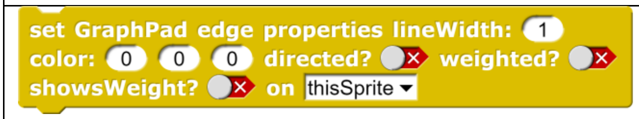
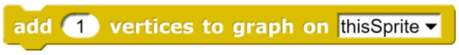
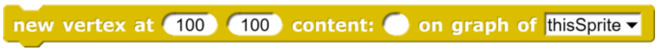
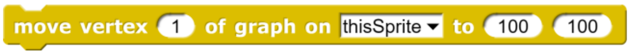
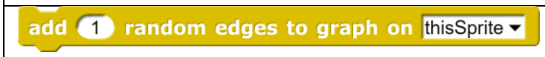
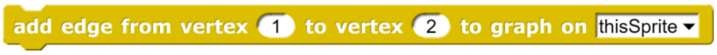
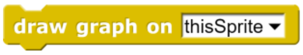
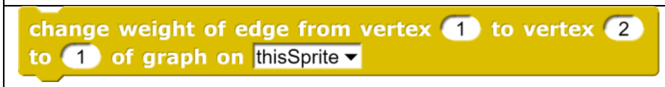


```
configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad ranges for x: 0 1000 y: -1 1
with border? ☒ of 0.1 pretty formatted? ✓
on thisSprite
add graph (sin of ( )) to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```

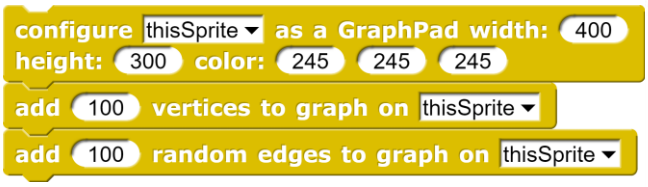


3.6 The GraphPad-Library (SciSnap!GraphPadLibrary.xml)

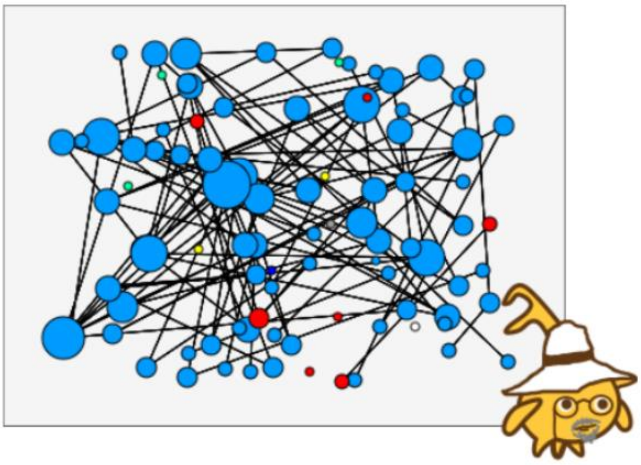
Graphs are one of the most powerful models in computer science. With their help, complex systems can be studied, especially the effect of the interconnection of numerous similar subsystems. However, the algorithms required for this, such as breadth-first search or routing, are themselves non-trivial, 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? thisSprite theStage anotherSprite	Returns "true" if the configuration is correct, otherwise "false".
 set GraphPadProperty costumeProperties ▾ of thisSprite ▾ to typeOfConfiguration typeOfData costumeProperties vertexProperties edgeProperties	Sets one of the properties in *myProperties* to the specified value.
 GraphPadProperty costumeProperties ▾ of thisSprite ▾ typeOfConfiguration typeOfData costumeProperties vertexProperties edgeProperties	Returns one of the properties in *myProperties*.
 set GraphPad costume properties width: 400 height: 300 color: 245 245 245 on thisSprite ▾	Sets the costume properties of the specified target.
 set GraphPad vertex properties minSize: 3 growing? ✓ showsContent? ✗ on thisSprite ▾	Sets the vertex (node) properties of the specified target.
 set GraphPad edge properties lineWidth: 1 color: 0 0 0 directed? ✗ weighted? ✗ showsWeight? ✗ on thisSprite ▾	Sets the edge properties of the specified target.
 add 1 vertices to graph on thisSprite ▾	Adds n vertices (nodes) to the graph at random positions.
 new vertex at 100 100 content: on graph of thisSprite ▾	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.

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

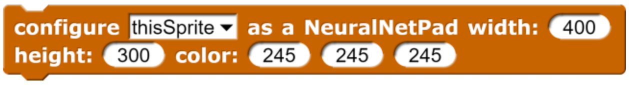
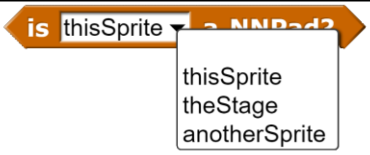
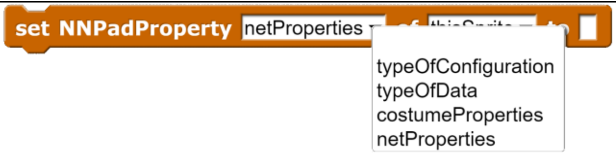
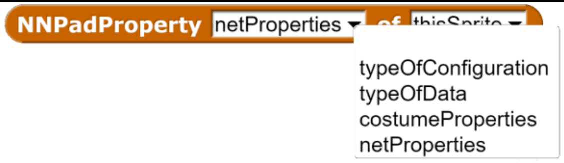
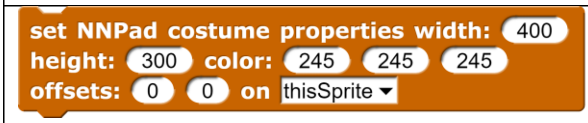
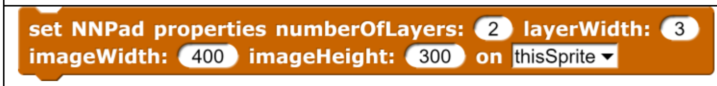
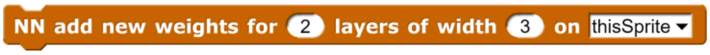
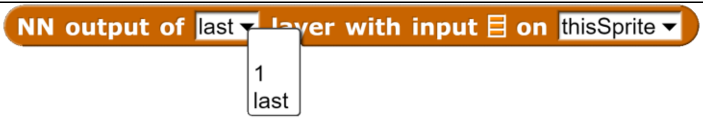
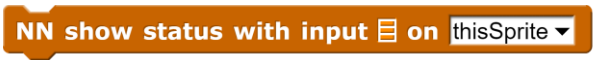
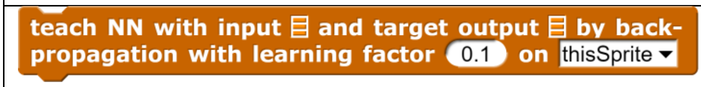


```
configure [thisSprite ▾] as a GraphPad width: (400)
height: (300) color: (245) (245) (245)
add (100) vertices to graph on [thisSprite ▾]
add (100) random edges to graph on [thisSprite ▾]
```



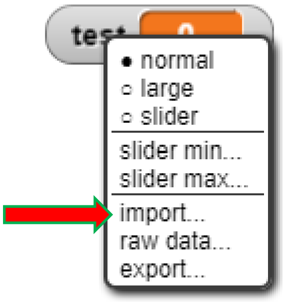
3.7 The NeuralNetPad-Library (SciSnap!NNPadLibrary.xml)

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.

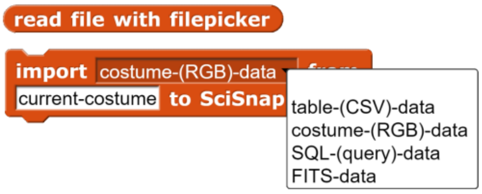
 configure thisSprite as a NeuralNetPad width: 400 height: 300 color: 245 245 245	Configures a sprite or the stage as *NNPad*. The name of "*anotherSprite*" must be specified if required. The command must be executed once before working with a sprite as *NNPad*. The target of the call takes a rectangular costume with the specified dimensions and colors.
 is thisSprite a NNPad? 	Returns "true" if the configuration is correct, otherwise "false".
 set NNPadProperty netProperties of thisSprite to typeOfConfiguration typeOfData costumeProperties netProperties	Sets one of the properties in *myProperties* to the specified value.
 NNPadProperty netProperties of thisSprite typeOfConfiguration typeOfData costumeProperties netProperties	Returns one of the properties in *myProperties*.
 set NNPad costume properties width: 400 height: 300 color: 245 245 245 offsets: 0 0 on thisSprite	Sets the costume properties of the specified target.
 set NNPad properties numberOfLayers: 2 layerWidth: 3 imageWidth: 400 imageHeight: 300 on thisSprite	Sets the layer properties of the specified target.
 NN add new weights for 2 layers of width 3 on thisSprite	Inserts random weights for an NN of the specified width and depth.
 NN output of last layer with input on thisSprite 	Returns the output of the nth layer of the NN for the specified input vector.
 NN show status with input on thisSprite	Displays the status of the network for the specified input vector in color-coded form.
 teach NN with input and target output by backpropagation with learning factor 0.1 on thisSprite	Trains the NN for the given input vector to achieve the given output vector.

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.

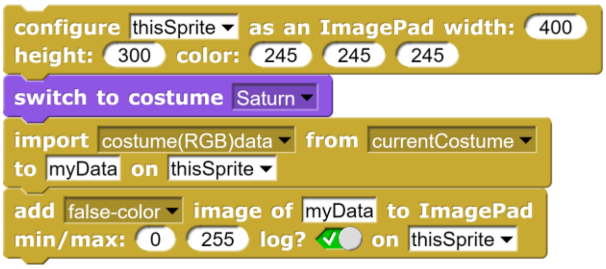


The essential task remains to assign this data to the *SciSnap!Data* variable and set the corresponding properties in *SciSnap!Properties*. This is done by the following block, which imports data from outside into the *SciSnap!Data* area. This can be image data, table data or the data of the current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

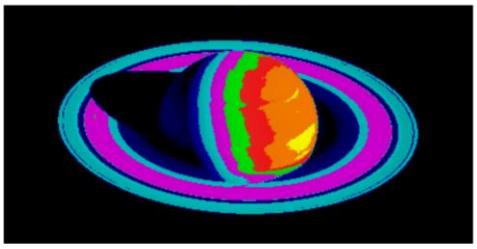

read file with filepicker
import costume-(RGB)-data from
current-costume to SciSnap
table-(CSV)-data
costume-(RGB)-data
SQL-(query)-data
FITS-data

Example: False color image

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

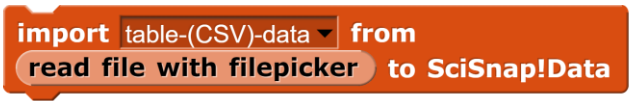


```
configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
switch to costume Saturn
import costume(RGB)data from currentCostume
to myData on thisSprite
add false-color image of myData to ImagePad
min/max: 0 255 log? ✓ on thisSprite
```

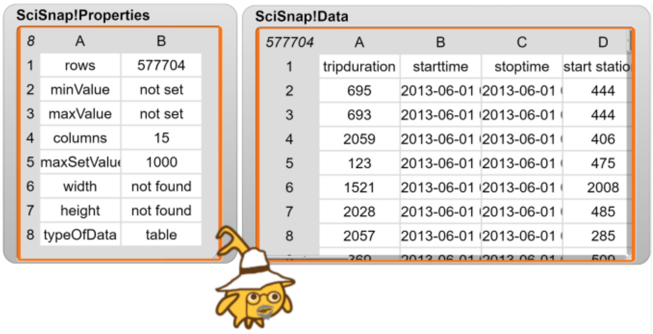


Example: CSV Import

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



```
import table-(CSV)-data from
read file with filepicker to SciSnap!Data
```



SciSnap!Properties

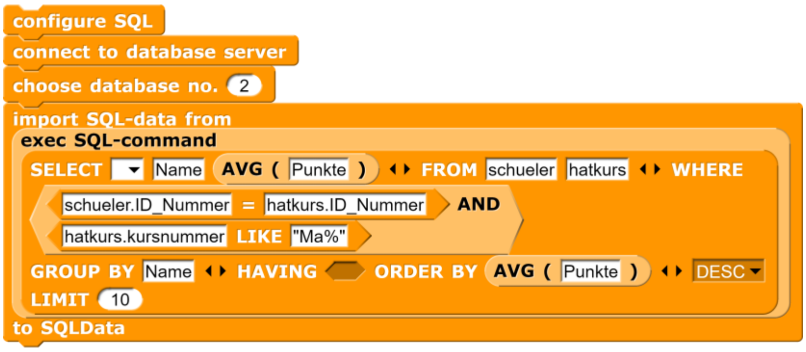
8	A	B
1	rows	577704
2	minValue	not set
3	maxValue	not set
4	columns	15
5	maxSetValu	1000
6	width	not found
7	height	not found
8	typeOfData	table

SciSnap!Data

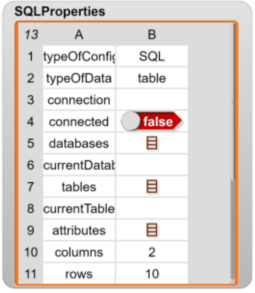
577704	A	B	C	D
1	tripduration	starttime	stoptime	start statio
2	695	2013-06-01	2013-06-01	444
3	693	2013-06-01	2013-06-01	444
4	2059	2013-06-01	2013-06-01	406
5	123	2013-06-01	2013-06-01	475
6	1521	2013-06-01	2013-06-01	2008
7	2028	2013-06-01	2013-06-01	485
8	2057	2013-06-01	2013-06-01	285

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 *SQLProproperties*.

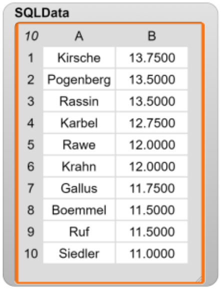


```
configure SQL
connect to database server
choose database no. (2)
import SQL-data from
  exec SQL-command
    SELECT [ ▾] Name (AVG ( Punkte )) ◂▸ FROM schueler hatkurs ◂▸ WHERE
      <<schueler.ID_Nummer = hatkurs.ID_Nummer> AND
       <hatkurs.kursnummer LIKE "Ma%">>
    GROUP BY Name ◂▸ HAVING < > ORDER BY (AVG ( Punkte )) ◂▸ DESC ▾
    LIMIT (10)
to SQLData
```



SQLProperties

13	A	B
1	typeOfConfig	SQL
2	typeOfData	table
3	connection	
4	connected	false
5	databases	☰
6	currentDatat	
7	tables	☰
8	currentTable	
9	attributes	☰
10	columns	2
11	rows	10

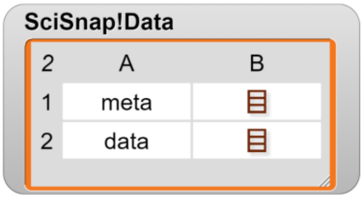


SQLData

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

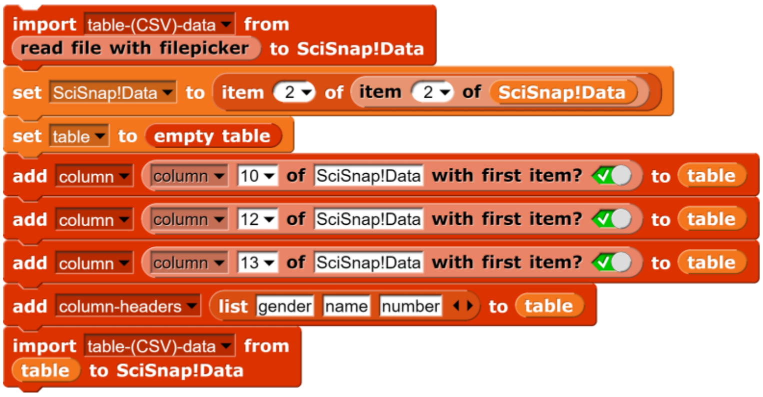
Example: JSON Import

Again, the easiest way is to simply "drop" a *JSON* file into the *Snap!* window. But it can also be done automatically. First of all, we look for interesting JSON data and of course choose the *statistics of baby names in New York City* for this - what else. The appropriate block for this is again *import <table data> from <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*.



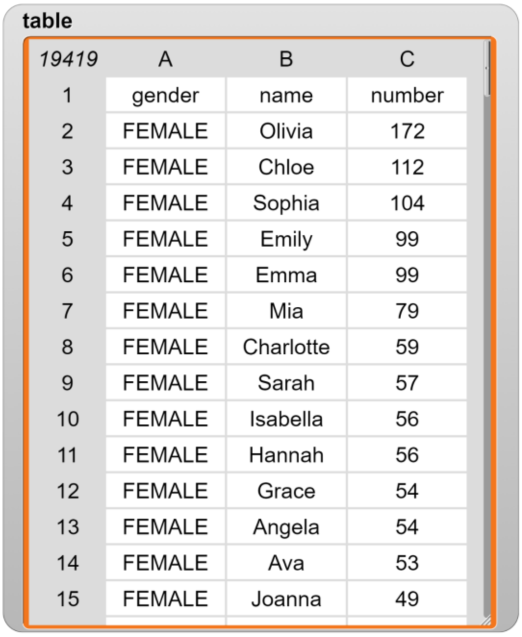
SciSnap!Data

2	A	B
1	meta	☰
2	data	☰



```
import table-(CSV)-data ▾ from
  (read file with filepicker) to SciSnap!Data
set SciSnap!Data ▾ to (item (2 ▾) of (item (2 ▾) of (SciSnap!Data)))
set table ▾ to (empty table)
add column ▾ (column ▾ (10 ▾) of SciSnap!Data with first item? ✓) to (table)
add column ▾ (column ▾ (12 ▾) of SciSnap!Data with first item? ✓) to (table)
add column ▾ (column ▾ (13 ▾) of SciSnap!Data with first item? ✓) to (table)
add column-headers ▾ (list gender name number ◂▸) to (table)
import table-(CSV)-data ▾ from
  (table) to SciSnap!Data
```

The result: 19419 baby names - Who would have thought it!

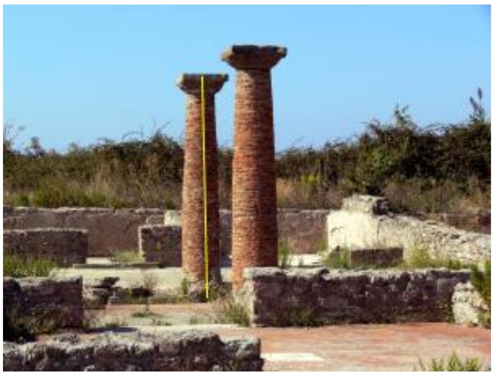


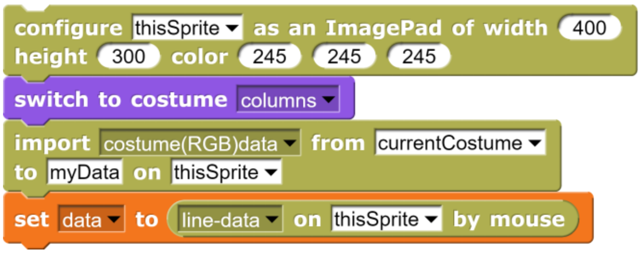
table

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

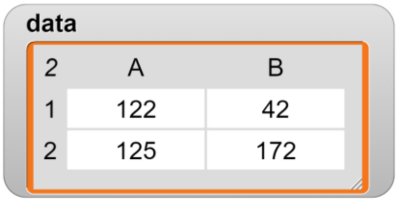
Example: Data import with the mouse

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).





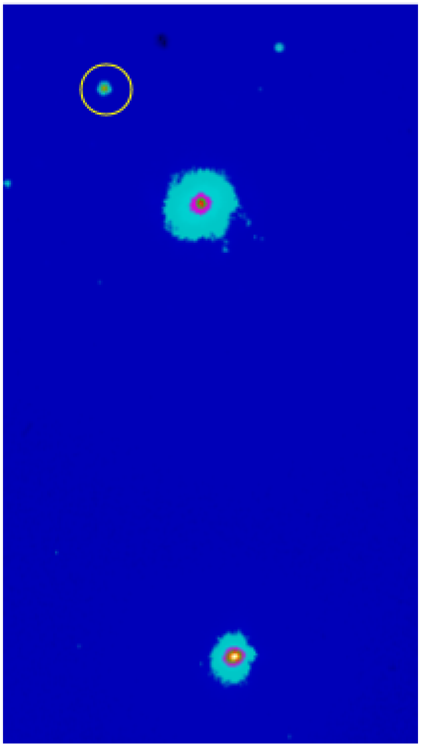
```
configure thisSprite as an ImagePad of width 400
height 300 color 245 245 245
switch to costume columns
import costume(RGB)data from currentCostume
to myData on thisSprite
set data to line-data on thisSprite by mouse
```

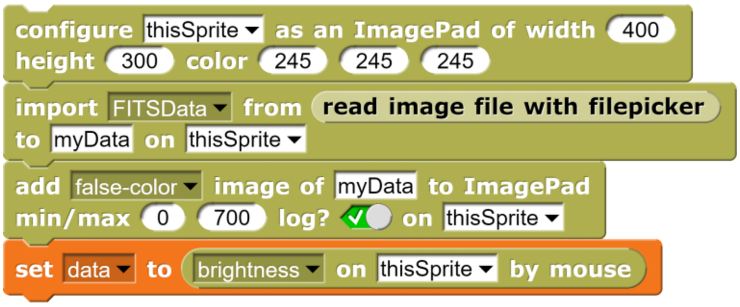


data

2	A	B
1	122	42
2	125	172

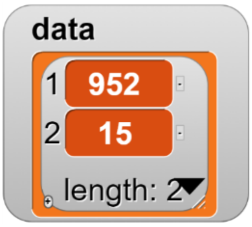
As a second example for measuring with the mouse we want to measure the total brightness inside a circle around a star photo (source: [HOU]).





```
configure thisSprite as an ImagePad of 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 0 700 log? ✓ on thisSprite
set data to brightness on thisSprite by mouse
```

We get the total brightness and the number of measured pixels.

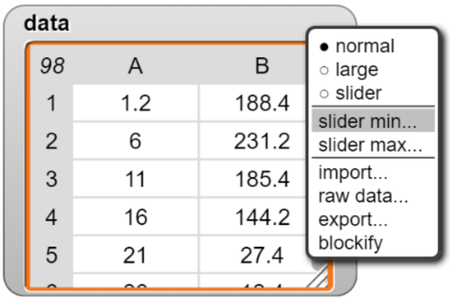


data

1	952
2	15

length: 2

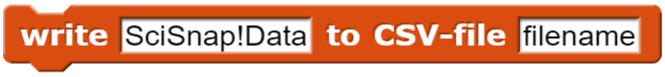
The export of data can again be done directly from a variable watcher.



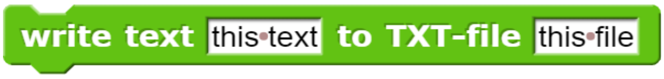
data

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

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.



write SciSnap!Data to CSV-file filename

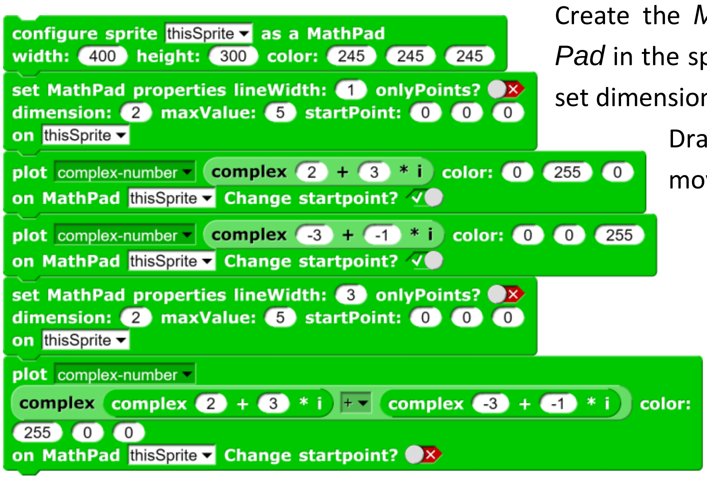


write text this text to TXT-file this file

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.



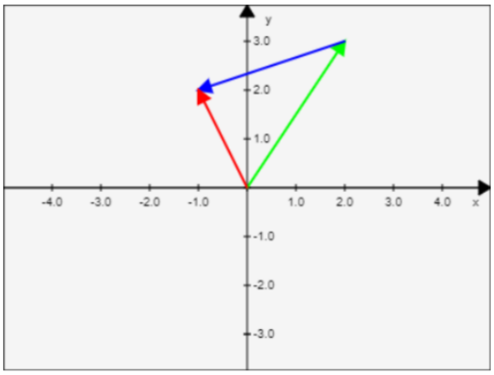
```
configure sprite thisSprite as a MathPad
width: 400 height: 300 color: 245 245 245
set MathPad properties lineWidth: 1 onlyPoints? ☒
dimension: 2 maxValue: 5 startPoint: 0 0 0
on thisSprite
plot complex-number complex 2 + 3 * i color: 0 255 0
on MathPad thisSprite Change startpoint? ✓
plot complex-number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

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.

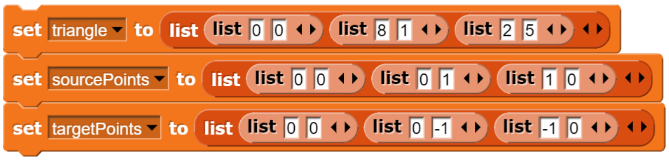
Draw the sum in red from the origin.



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

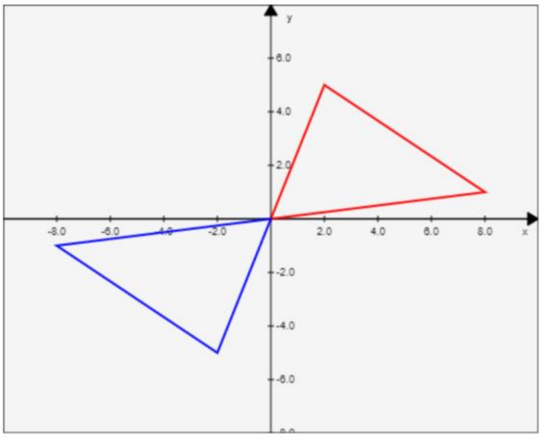
5.2 Affine transformation of a triangle in R²

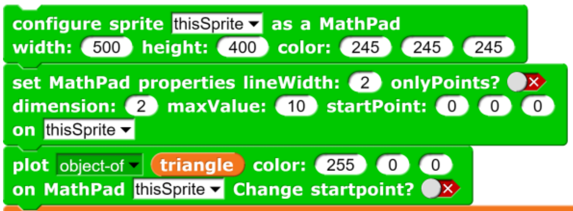
We define a triangle by its location vectors. Then we define three points in the plane and three points to map them to:



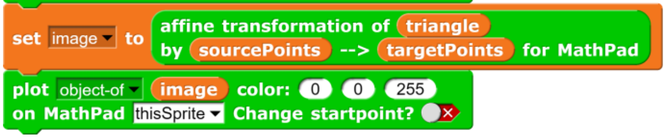
```
set triangle to list (list 0 0) (list 8 1) (list 2 5)
set sourcePoints to list (list 0 0) (list 0 1) (list 1 0)
set targetPoints to list (list 0 0) (list 0 -1) (list -1 0)
```

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.





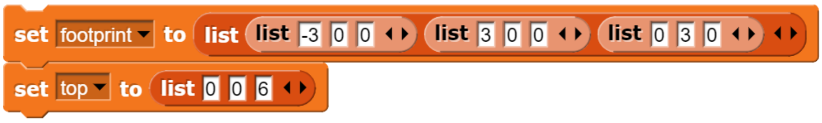
```
configure sprite thisSprite as a MathPad
width: 500 height: 400 color: 245 245 245
set MathPad properties lineWidth: 2 onlyPoints?
dimension: 2 maxValue: 10 startPoint: 0 0 0
on thisSprite
plot object-of triangle color: 255 0 0
on MathPad thisSprite Change startpoint?
```



```
set image to affine transformation of triangle
             by sourcePoints --> targetPoints for MathPad
plot object-of image color: 0 0 255
on MathPad thisSprite Change startpoint?
```

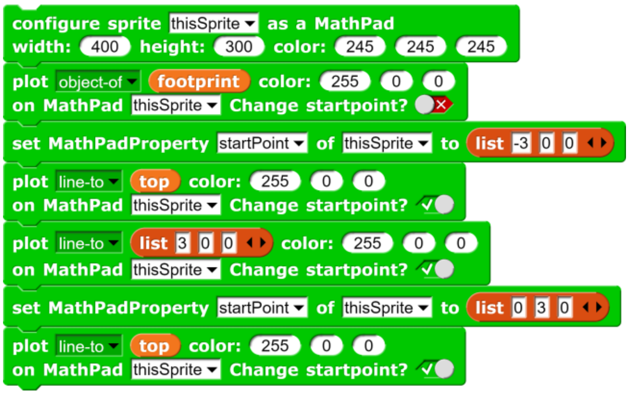
5.3 Rotation of a pyramid in R³

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

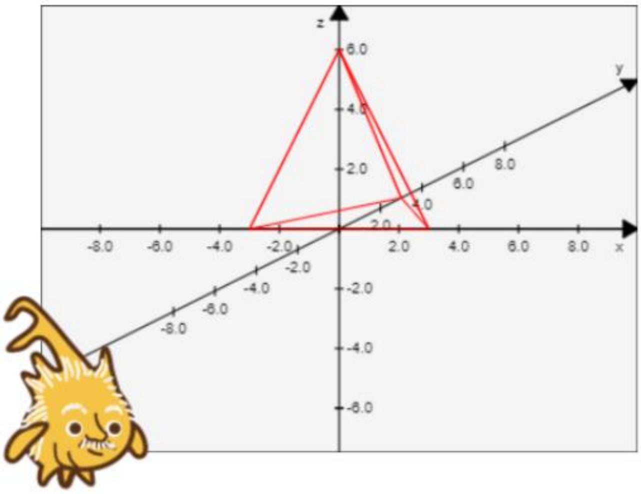


```
set footprint to list (list -3 0 0) (list 3 0 0) (list 0 3 0)
set top to list 0 0 6
```

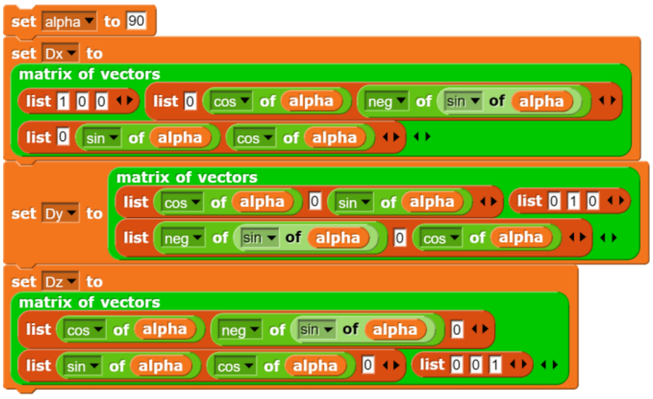
We have them drawn by first drawing the base area, then lines from its corners to the top.



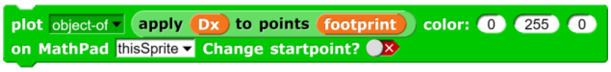
```
configure sprite thisSprite as a MathPad
width: 400 height: 300 color: 245 245 245
plot object-of footprint color: 255 0 0
on MathPad thisSprite Change startpoint? ☒
set MathPadProperty startPoint of thisSprite to list -3 0 0
plot line-to top color: 255 0 0
on MathPad thisSprite Change startpoint?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



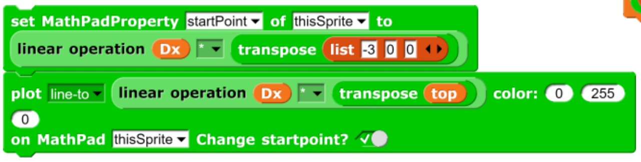
For rotations around the axes we need the three rotation matrices D_x, D_y and D_z. For a rotation around the x-axis by 90° we can apply the matrix directly to the base surface. Then we let draw the side lines to the rotated point by multiplying the rotation matrix with the transposed location vectors of the points.



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

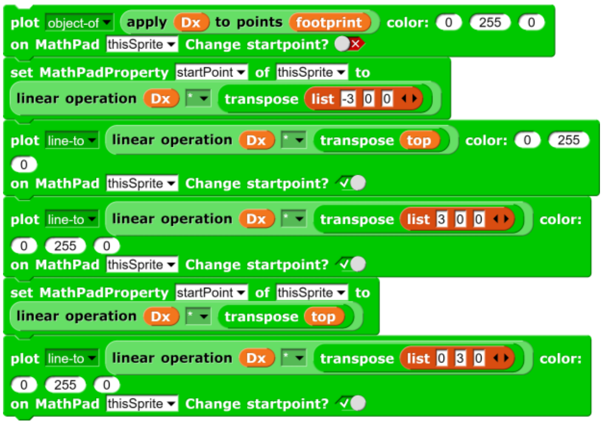


```
plot object-of (apply Dx to points footprint) color: 0 255 0
on MathPad thisSprite Change startpoint? ☒
```

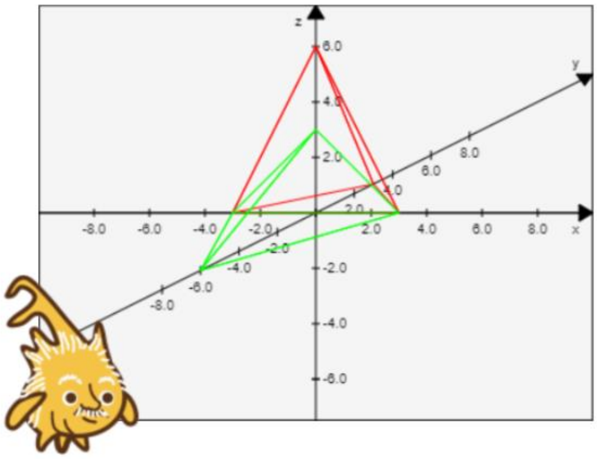


```
set MathPadProperty startPoint of thisSprite to
  linear operation Dx · transpose list -3 0 0
plot line-to linear operation Dx · transpose top color: 0 255
0
on MathPad thisSprite Change startpoint? ✓
```

So in total:

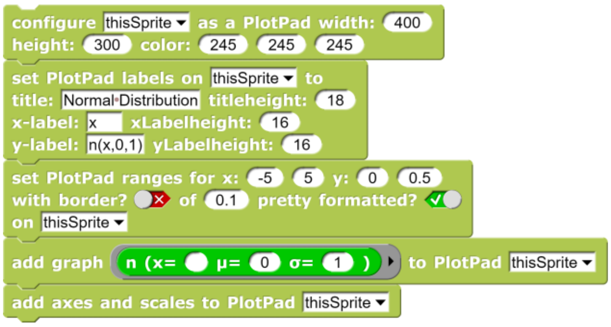


```
plot object-of (apply Dx to points footprint) color: 0 255 0
on MathPad thisSprite Change startpoint? ☒
set MathPadProperty startPoint of thisSprite to
  linear operation Dx · transpose list -3 0 0
plot line-to linear operation Dx · transpose top color: 0 255
0
on MathPad thisSprite Change startpoint?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

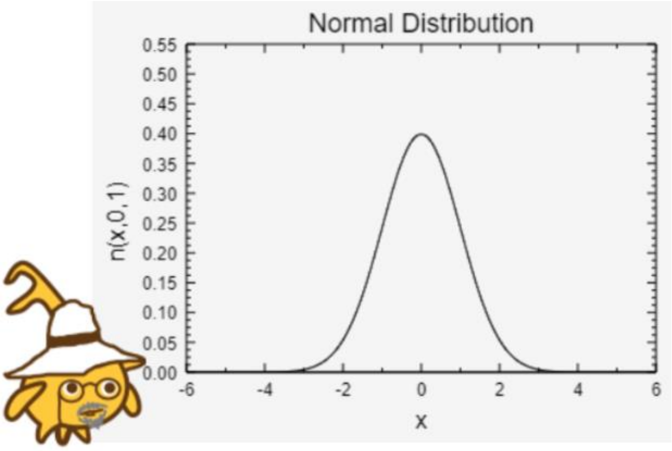


5.4 Graph of normal distribution

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

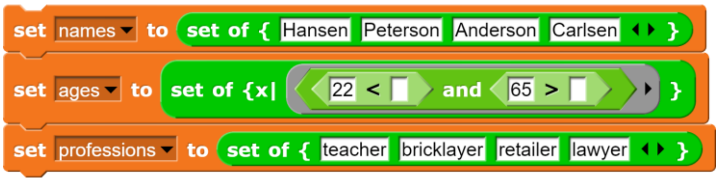


```
configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on thisSprite to
title: Normal Distribution titleheight: 18
x-label: x xLabelheight: 16
y-label: n(x,0,1) yLabelheight: 16
set PlotPad ranges for x: -5 5 y: 0 0.5
with border? ☒ of 0.1 pretty formatted? ✓
on thisSprite
add graph n (x= μ= 0 σ= 1 ) to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```



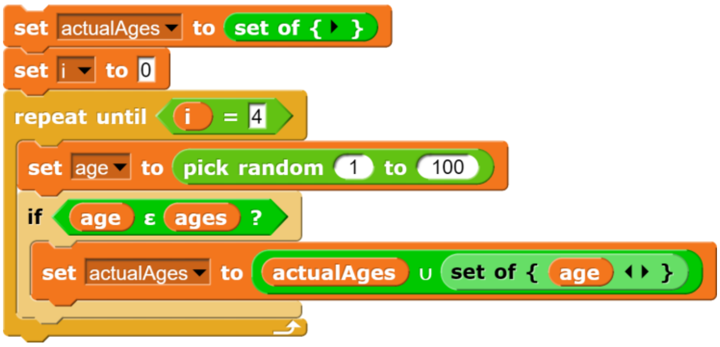
5.5 Cartesian product of three sets

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



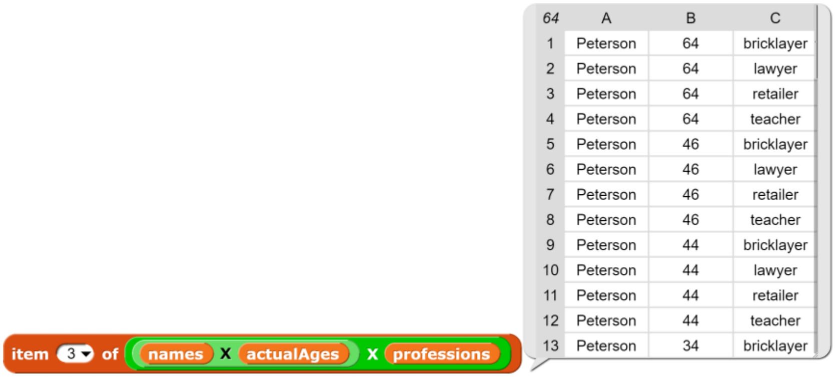
```
set names to set of { Hansen Peterson Anderson Carlsen ◂▸ }
set ages to set of {x| 22 < □ and 65 > □ ▸ }
set professions to set of { teacher bricklayer retailer lawyer ◂▸ }
```

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 ∪ set of { age ◂▸ }
```

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



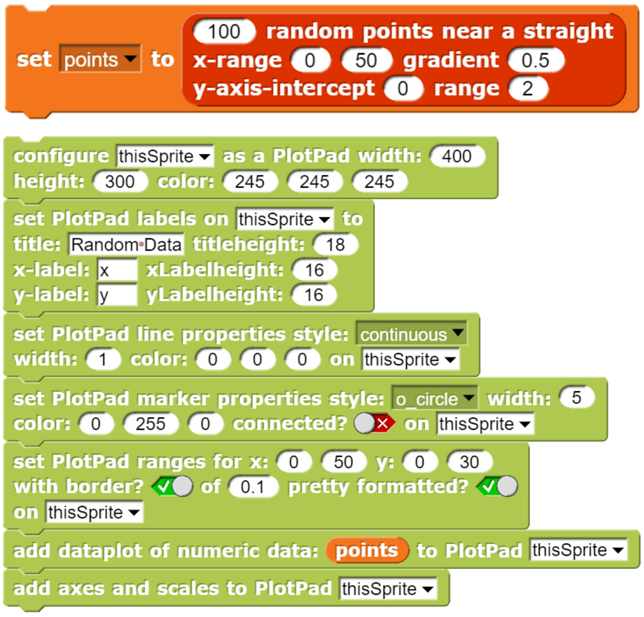
```
item 3 of names X actualAges X professions
```

64	A	B	C
1	Peterson	64	bricklayer
2	Peterson	64	lawyer
3	Peterson	64	retailer
4	Peterson	64	teacher
5	Peterson	46	bricklayer
6	Peterson	46	lawyer
7	Peterson	46	retailer
8	Peterson	46	teacher
9	Peterson	44	bricklayer
10	Peterson	44	lawyer
11	Peterson	44	retailer
12	Peterson	44	teacher
13	Peterson	34	bricklayer

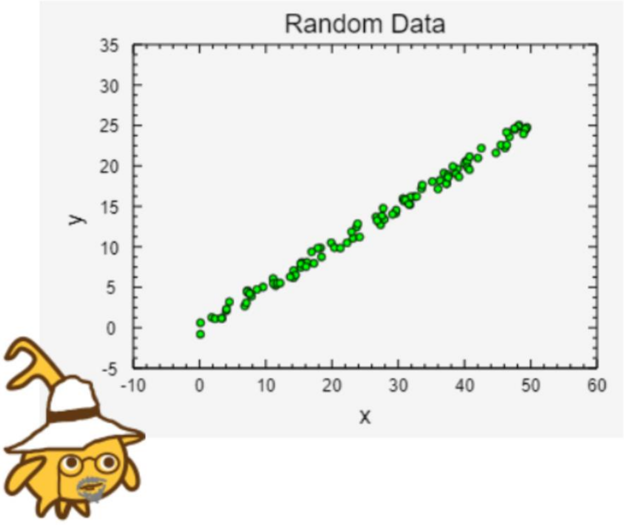
And thus nothing stands in the way of a transition to the topic of "relational databases", for example.

5.6 Representation of a set of points and the regression line

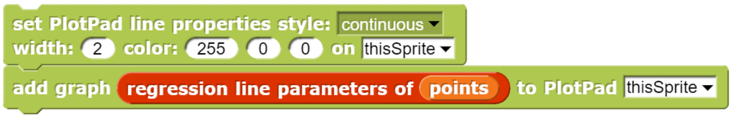
Using the *SciSnap!DataLibrary*, we create 100 random points scattering around a straight line with slope *m=0.5* and *intercept b=0*. We plot the obtained points in a diagram.



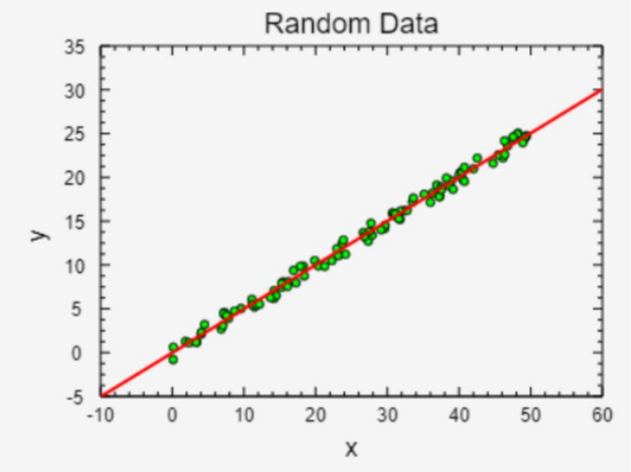
```
set points ▾ to (100 random points near a straight x-range 0 50 gradient 0.5 y-axis-intercept 0 range 2)
configure thisSprite ▾ as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on thisSprite ▾ to
title: Random Data titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16
set PlotPad line properties style: continuous ▾
width: 1 color: 0 0 0 on thisSprite ▾
set PlotPad marker properties style: o_circle ▾ width: 5
color: 0 255 0 connected? ☒ on thisSprite ▾
set PlotPad ranges for x: 0 50 y: 0 30
with border? ✓ of 0.1 pretty formatted? ✓
on thisSprite ▾
add dataplot of numeric data: (points) to PlotPad thisSprite ▾
add axes and scales to PlotPad thisSprite ▾
```



In addition, the regression line is also drawn.



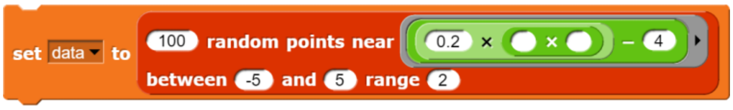
```
set PlotPad line properties style: continuous ▾
width: 2 color: 255 0 0 on thisSprite ▾
add graph (regression line parameters of (points)) to PlotPad thisSprite ▾
```



5.7 Interpolation polynomial through n points

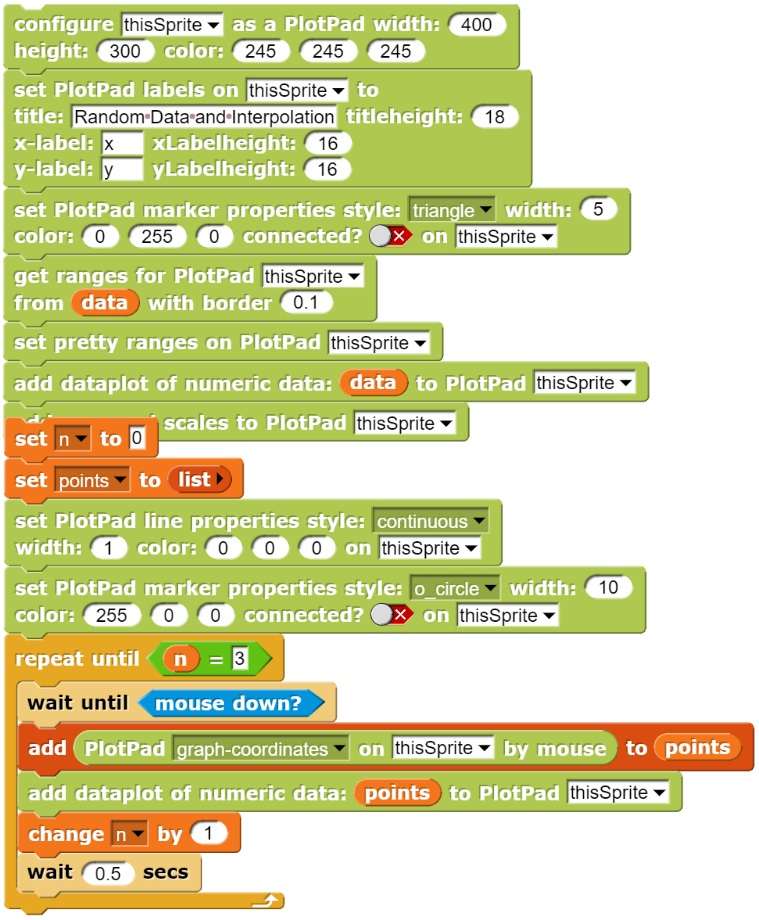
We want to generate a data set of 100 points that scatter around a given function. We display these points in a diagram. We then select three points by clicking on the corresponding locations with the mouse and placing a red marker there. After that we draw an interpolation polynomial in red through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

First of all the random points:

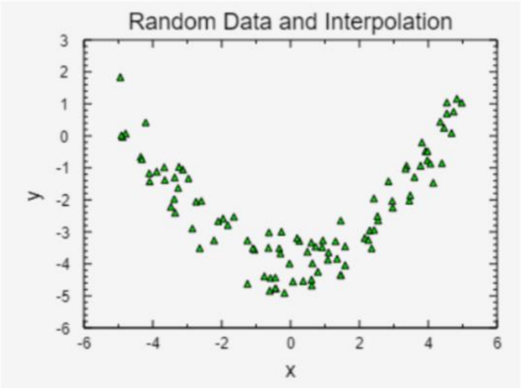


```
set data to (100 random points near ((0.2 × ( ) × ( )) − 4) between -5 and 5 range 2)
```

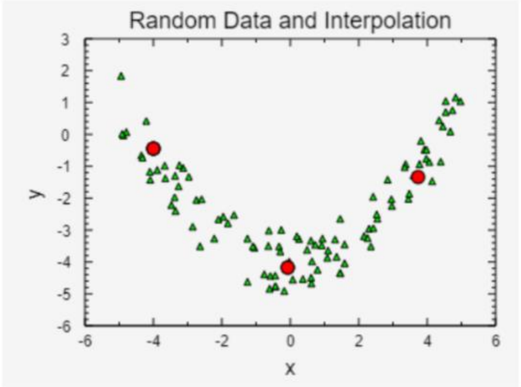
After that we configure the current sprite to the *PlotPad* and draw the point set.



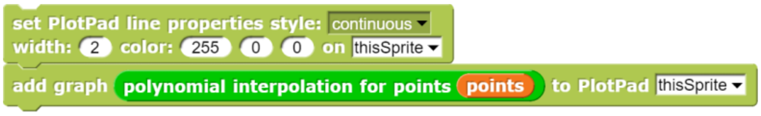
```
configure thisSprite as a PlotPad width: 400 height: 300 color: 245 245 245
set PlotPad labels on thisSprite to
  title: Random Data and Interpolation titleheight: 18
  x-label: x xLabelheight: 16
  y-label: y yLabelheight: 16
set PlotPad marker properties style: triangle width: 5 color: 0 255 0 connected? ✗ on thisSprite
get ranges for PlotPad thisSprite from (data) with border 0.1
set pretty ranges on PlotPad thisSprite
add dataplot of numeric data: (data) to PlotPad thisSprite
scales to PlotPad thisSprite
set n to 0
set points to (list)
set PlotPad line properties style: continuous width: 1 color: 0 0 0 on thisSprite
set PlotPad marker properties style: o_circle width: 10 color: 255 0 0 connected? ✗ on thisSprite
repeat until ((n) = 3)
  wait until (mouse down?)
  add (PlotPad graph-coordinates on thisSprite by mouse) to (points)
  add dataplot of numeric data: (points) to PlotPad thisSprite
  change n by 1
  wait 0.5 secs
```



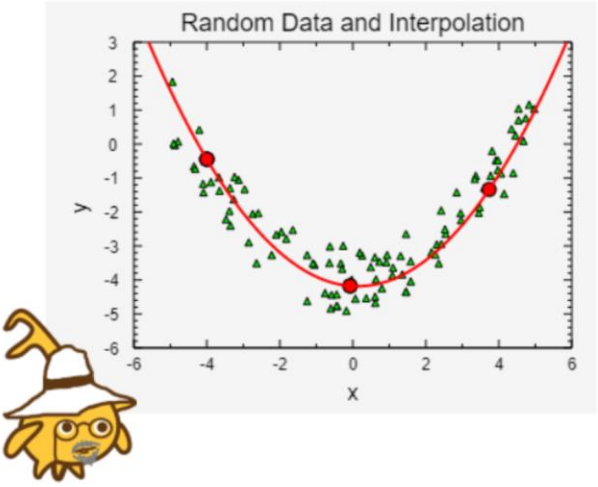
Now we select the three points and plot them in the diagram right away.



And last we add the interpolation polynomial.



```
set PlotPad line properties style: continuous width: 2 color: 255 0 0 on thisSprite
add graph (polynomial interpolation for points (points)) to PlotPad thisSprite
```



<u>Tasks</u>:

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.

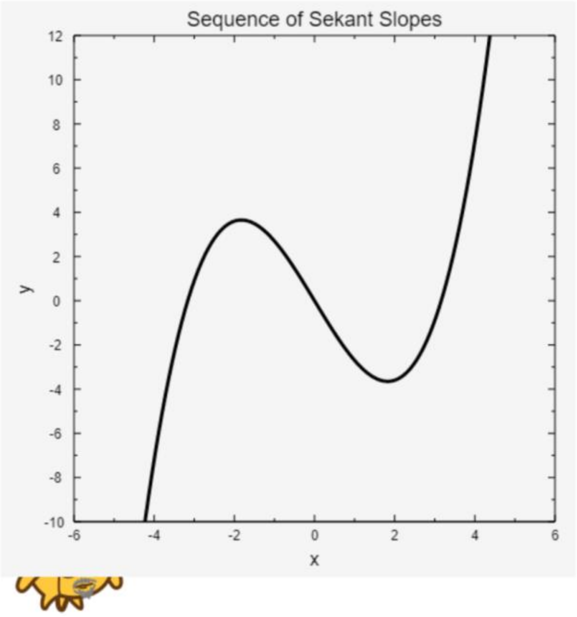
2. a: Experiment with the number of points you select. Do the results get better when you select more points?

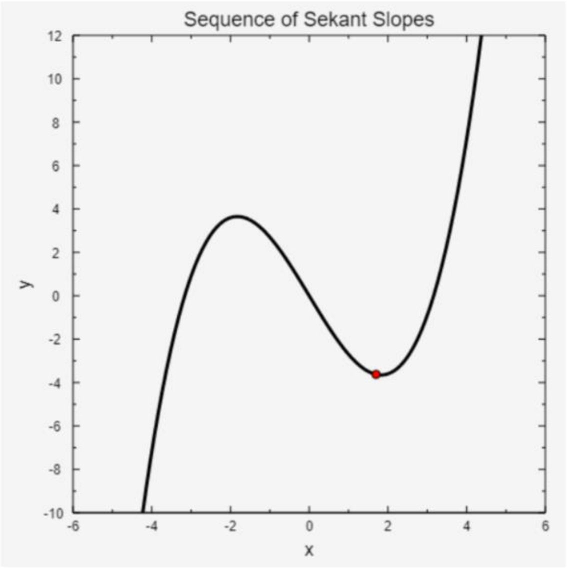
   b: Generate "point clouds" that scatter around non-holistic function graphs (trigonometric, ...). Can you also describe these by interpolation polynomials?

   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

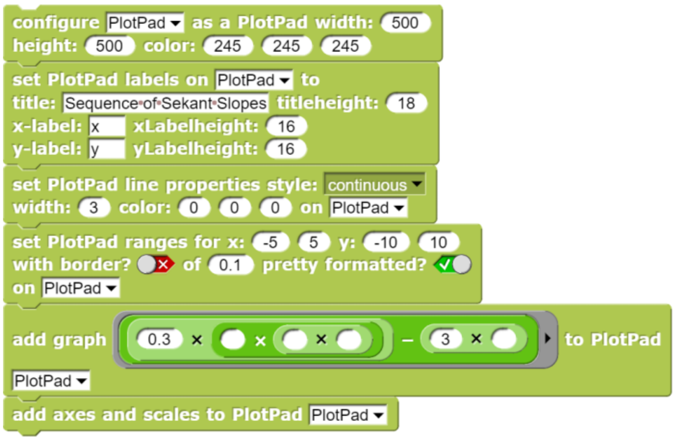
We want to show that a sequence of secant slopes converges against the slope of the tangent line at a point. To do this, we configure a sprite called *PlotPad* as *PlotPad* and draw the graph of a function, here: $y = 0{,}3 \cdot x^3 - 3 \cdot x$.





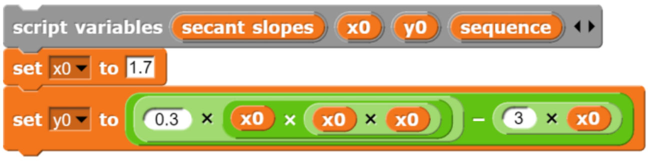
The rest is just as simple: we have the secant slopes calculated ...

... und die Sekanten in Farbabstufungen zeichnen.



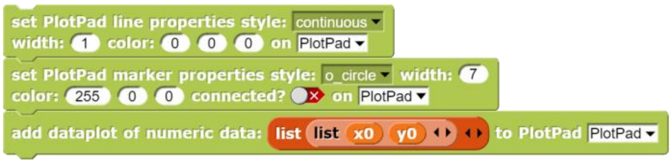
```
configure PlotPad as a PlotPad width: 500
height: 500 color: 245 245 245
set PlotPad labels on PlotPad to
title: Sequence of Sekant Slopes titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16
set PlotPad line properties style: continuous
width: 3 color: 0 0 0 on PlotPad
set PlotPad ranges for x: -5 5 y: -10 10
with border? ☒ of 0.1 pretty formatted? ✓
on PlotPad
add graph (0.3 × ( ○ × ( ○ × ○ )) − (3 × ○ )) to PlotPad
PlotPad
add axes and scales to PlotPad PlotPad
```

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:



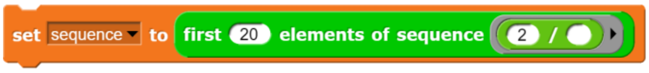
```
script variables secant slopes x0 y0 sequence
set x0 to 1.7
set y0 to (0.3 × (x0 × (x0 × x0)) − (3 × x0))
```

We may as well draw it in.

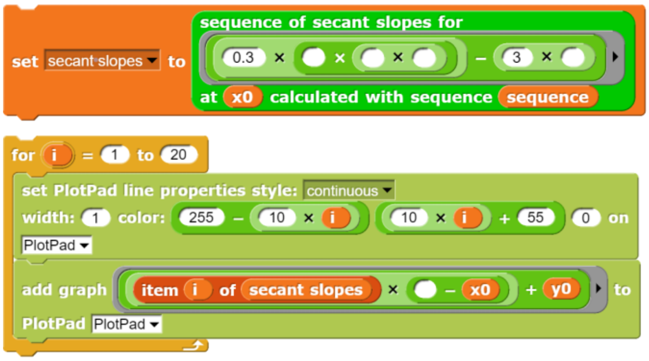


```
set PlotPad line properties style: continuous
width: 1 color: 0 0 0 on PlotPad
set PlotPad marker properties style: o_circle width: 7
color: 255 0 0 connected? ☒ on PlotPad
add dataplot of numeric data: list (list x0 y0) to PlotPad PlotPad
```

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

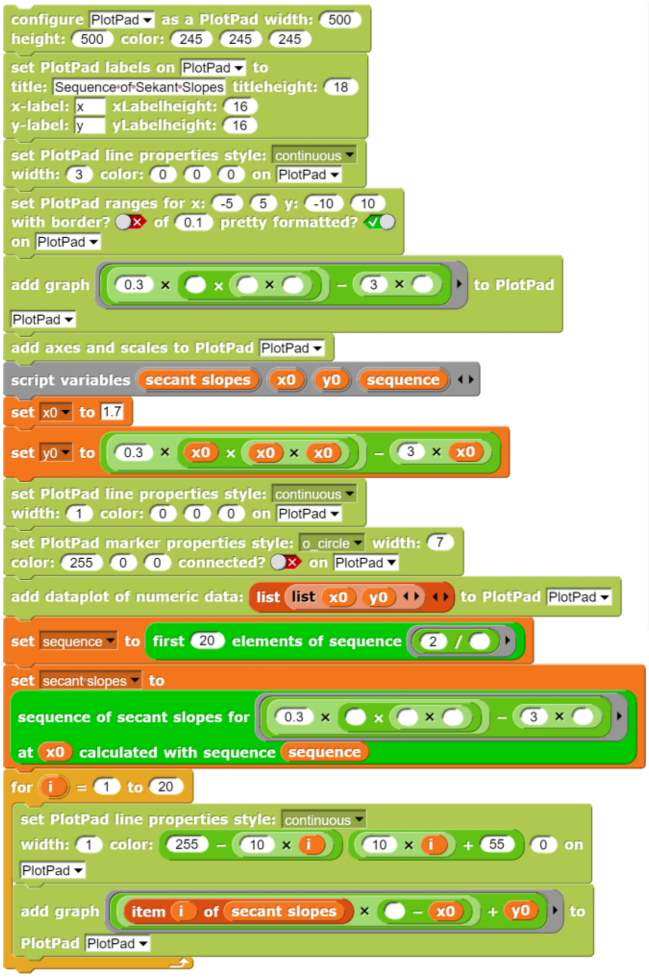


```
set sequence to first 20 elements of sequence (2 / ○ )
```

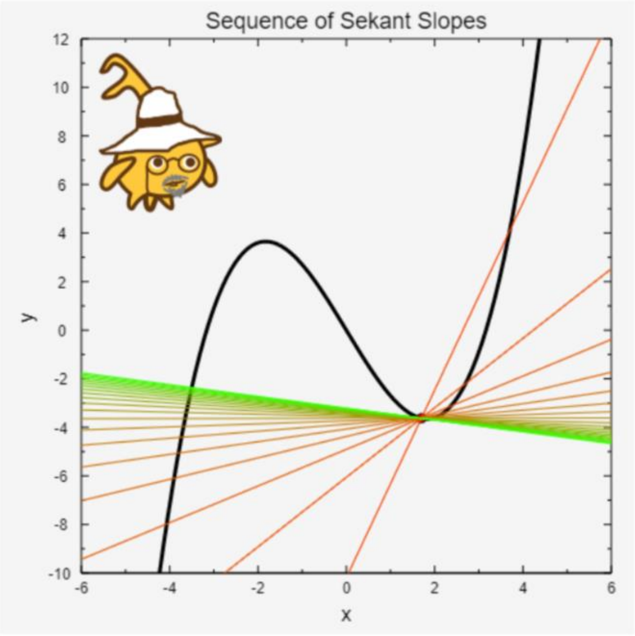


```
set secant slopes to sequence of secant slopes for
    (0.3 × ( ○ × ( ○ × ○ )) − (3 × ○ ))
    at x0 calculated with sequence sequence
for i = 1 to 20
    set PlotPad line properties style: continuous
    width: 1 color: (255 − (10 × i)) ((10 × i) + 55) 0 on
    PlotPad
    add graph ((item i of secant slopes) × ( ○ − x0) + y0) to
    PlotPad PlotPad
```

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



```
configure PlotPad as a PlotPad width: 500
height: 500 color: 245 245 245
set PlotPad labels on PlotPad to
title: Sequence of Sekant Slopes titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16
set PlotPad line properties style: continuous
width: 3 color: 0 0 0 on PlotPad
set PlotPad ranges for x: -5 5 y: -10 10
with border? ☒ of 0.1 pretty formatted? ✓
on PlotPad
add graph ((0.3 × ( × ( × ))) − (3 × )) to PlotPad
PlotPad
add axes and scales to PlotPad PlotPad
script variables secant slopes x0 y0 sequence
set x0 to 1.7
set y0 to (0.3 × (x0 × (x0 × x0))) − (3 × x0)
set PlotPad line properties style: continuous
width: 1 color: 0 0 0 on PlotPad
set PlotPad marker properties style: o_circle width: 7
color: 255 0 0 connected? ☒ on PlotPad
add dataplot of numeric data: list (list x0 y0) to PlotPad PlotPad
set sequence to first 20 elements of sequence (2 / )
set secant slopes to
  sequence of secant slopes for ((0.3 × ( × ( × ))) − (3 × ))
  at x0 calculated with sequence sequence
for i = 1 to 20
  set PlotPad line properties style: continuous
  width: 1 color: (255 − (10 × i)) ((10 × i) + 55) 0 on
  PlotPad
  add graph ((item i of secant slopes) × ( − x0)) + y0 to
  PlotPad PlotPad
```



<u>Tasks</u>:

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

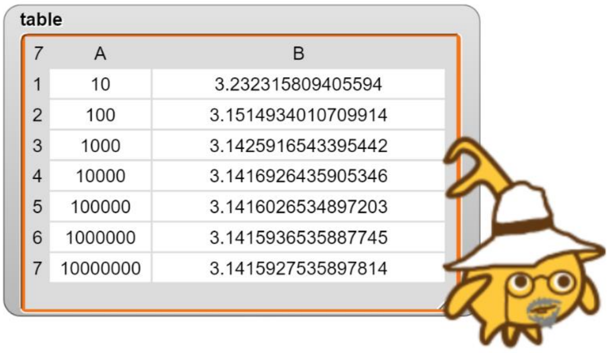
We want to approximate some of the common mathematical constants and functions via series expansions - and also try out how long you actually have to calculate to get good results.

Let's start with π. In a compendium of formulas [18] or on Wikipedia [19] we can find a formula for calculating π: the *Leibniz-series*: $\frac{\pi}{4} = \sum_{i=0}^{n} \frac{(-1)^i}{(2 \cdot i+1)}$

We can implement them directly in *SciSnap!*.

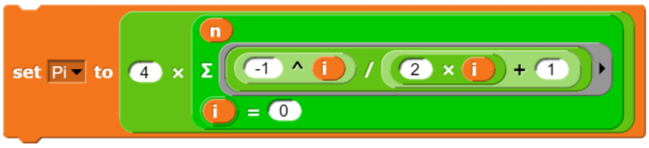
But when is the result actually "good"?

To answer this question we create a table.

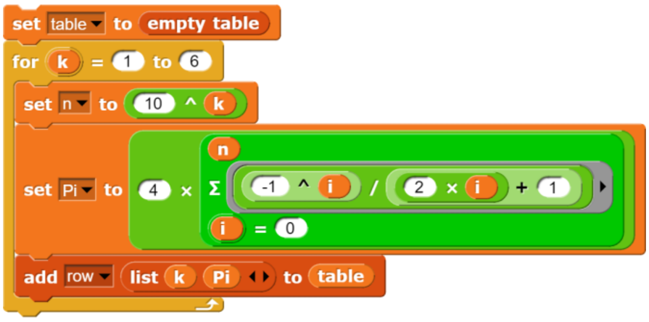


table

7	A	B
1	10	3.232315809405594
2	100	3.1514934010709914
3	1000	3.1425916543395442
4	10000	3.1416926435905346
5	100000	3.1416026534897203
6	1000000	3.1415936535887745
7	10000000	3.1415927535897814



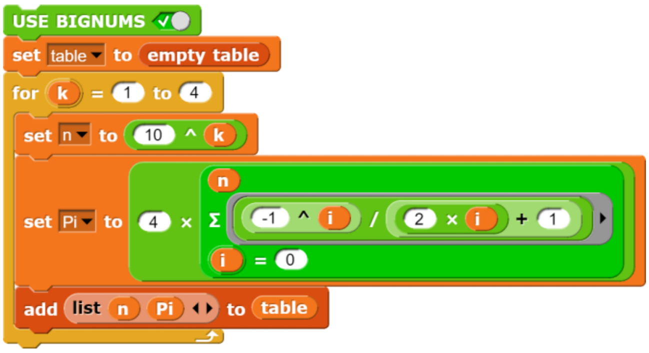
```
set Pi to 4 × Σ n ((-1 ^ i) / ((2 × i) + 1)) i = 0
```



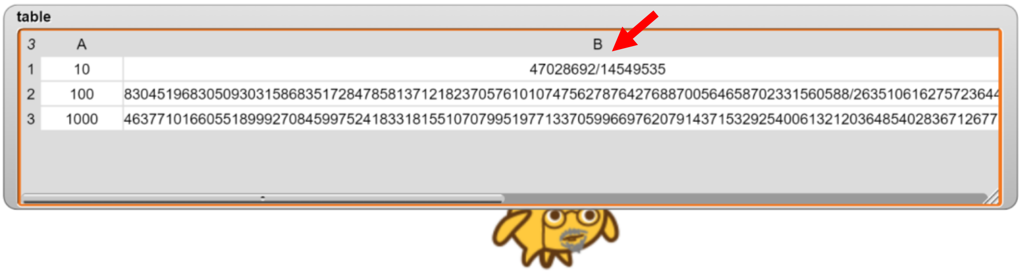
```
set table to empty table
for k = 1 to 6
    set n to 10 ^ k
    set Pi to 4 × Σ n ((-1 ^ i) / ((2 × i) + 1)) i = 0
    add row list k Pi to table
```

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 π.



```
USE BIGNUMS ✓
set table to empty table
for k = 1 to 4
    set n to 10 ^ k
    set Pi to 4 × Σ n ((-1 ^ i) / ((2 × i) + 1)) i = 0
    add list n Pi to table
```



table

3	A	B
1	10	47028692/14549535
2	100	8304519683050930315868351728478581371218237057610107475627876427688700564658702331560588/263510616275723644
3	1000	4637710166055189992708459975241833181551070799519771337059966976207914371532925400613212036485402836712677

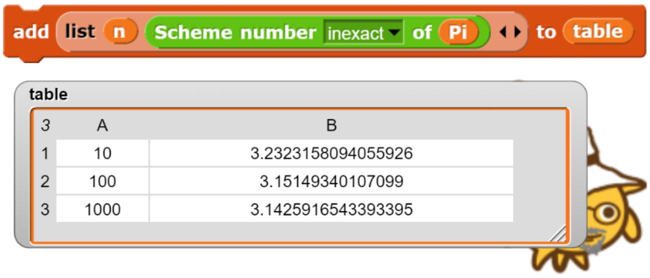
After a long search, we discover the fraction bar in the middle, although it is no longer visible in the second line. *Scheme numbers* are exact fractions and not floating point numbers.

[18] Ask your grandpa what it is. 😉

[19] https://en.wikipedia.org/wiki/Leibniz_formula_for_π

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! 😉



```
add list n Scheme number inexact of Pi ◂▸ to table
```

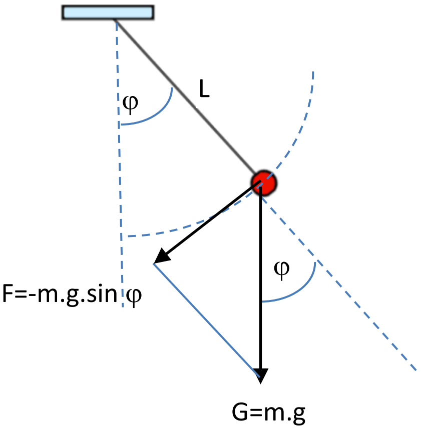
table

3	A	B
1	10	3.2323158094055926
2	100	3.15149340107099
3	1000	3.1425916543393395

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



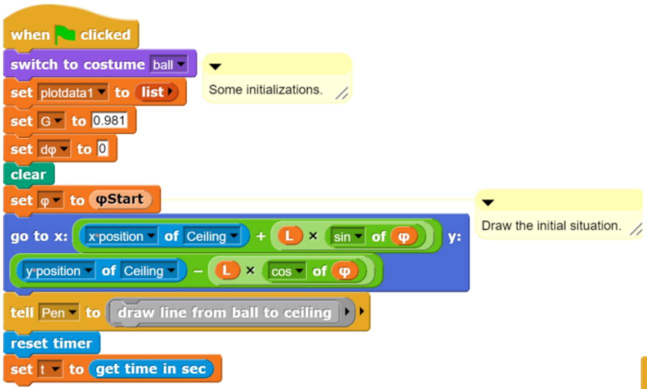
A very illustrative application of series expansions is the simulation of a mathematical pendulum, i.e. a string pendulum. Usually, one works there with the approximation that for small angles the value of the sine (in radians) corresponds approximately to the value of its argument - i.e. one breaks 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

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

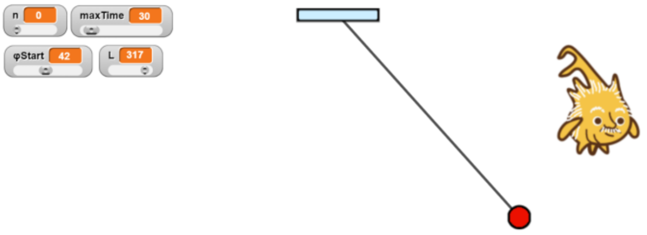
For small angles the approximate $\sin\varphi \approx \varphi$ is valid and thus

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

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.

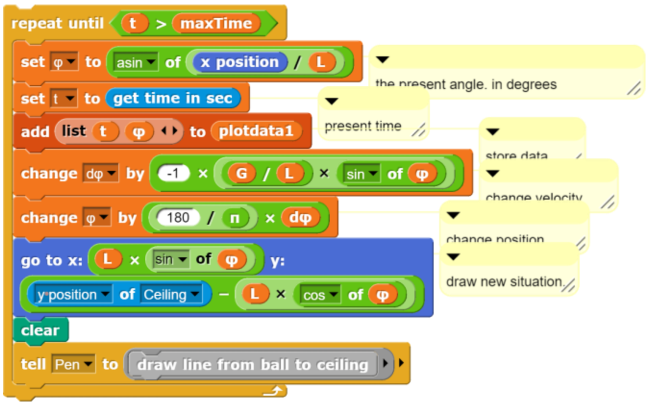


```
when [flag] clicked
switch to costume ball
set plotdata1 to list              Some initializations.
set G to 0.981
set dφ to 0
clear
set φ to φStart
go to x: (x position of Ceiling) + (L × sin of φ)  y:      Draw the initial situation.
    (y position of Ceiling) − (L × cos of φ)
tell Pen to draw line from ball to ceiling
reset timer
set t to get time in sec
```



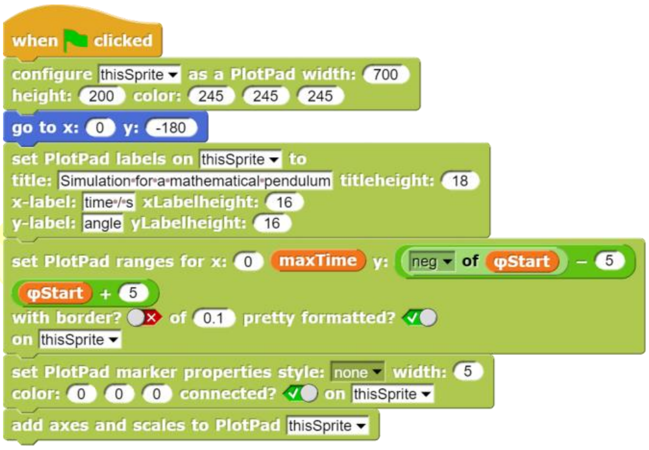
```
n 0    maxTime 30
φStart 42    L 317
```

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.

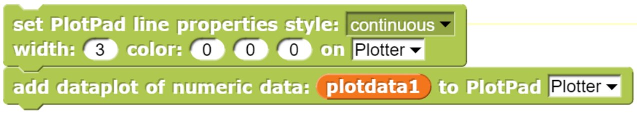


```
repeat until (t > maxTime)
    set φ to asin of (x position / L)                  the present angle. in degrees
    set t to get time in sec                           present time
    add list t φ to plotdata1                          store data
    change dφ by -1 × (G / L) × sin of φ               change velocity
    change φ by 180 / n × dφ                           change position
    go to x: L × sin of φ  y:                          draw new situation.
        (y position of Ceiling) − (L × cos of φ)
    clear
    tell Pen to draw line from ball to ceiling
```

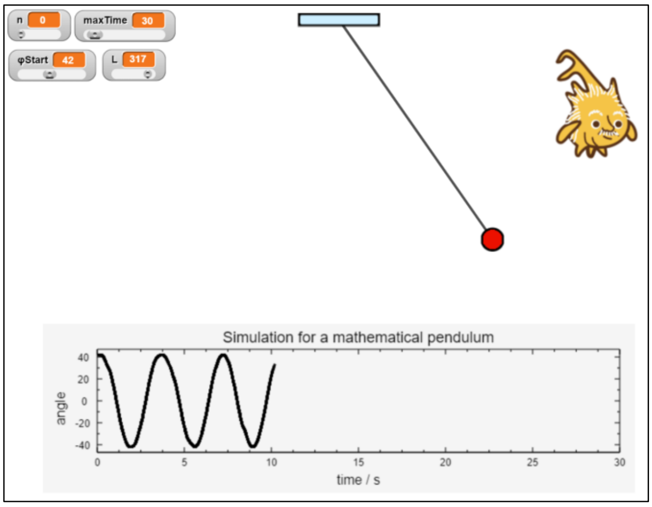
To this arrangement we add another sprite as PlotPad: the *Plotter*. It is fast enough to display the current data in real time. Therefore we insert the block for it into the simulation loop of the pendulum.



```
when 🏳 clicked
configure thisSprite as a PlotPad width: 700 height: 200 color: 245 245 245
go to x: 0 y: -180
set PlotPad labels on thisSprite to
title: Simulation for a mathematical pendulum titleheight: 18
x-label: time / s xLabelheight: 16
y-label: angle yLabelheight: 16
set PlotPad ranges for x: 0 maxTime y: (neg of φStart) − 5
φStart + 5
with border? ✗ of 0.1 pretty formatted? ✓
on thisSprite
set PlotPad marker properties style: none width: 5
color: 0 0 0 connected? ✓ on thisSprite
add axes and scales to PlotPad thisSprite
```

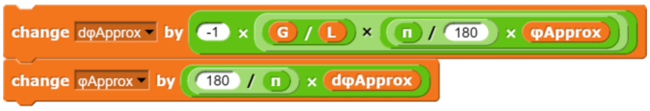


```
set PlotPad line properties style: continuous
width: 3 color: 0 0 0 on Plotter
add dataplot of numeric data: plotdata1 to PlotPad Plotter
```

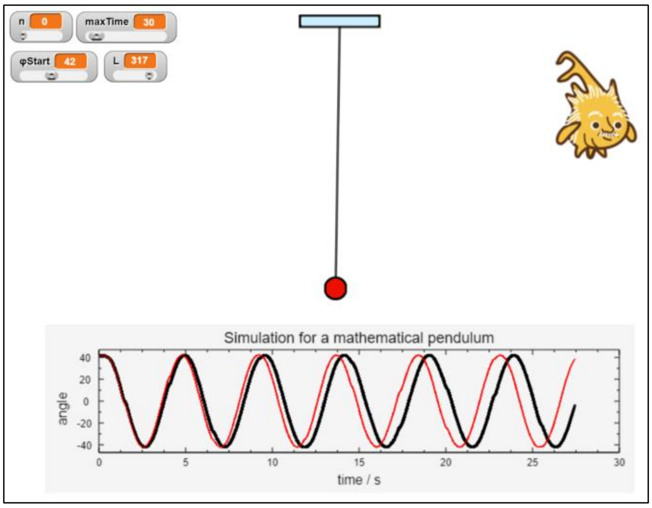


Alberto is visibly thrilled that this is working out so well!

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



```
change dφApprox by (-1 × (G / L) × ((π / 180) × φApprox))
change φApprox by (180 / π) × dφApprox
```



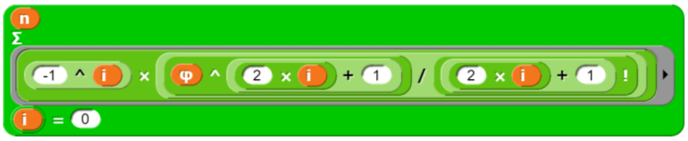
You can see that the approximation is quite good at the beginning, but diverges with the real-time values as time goes on.

This can be done better!

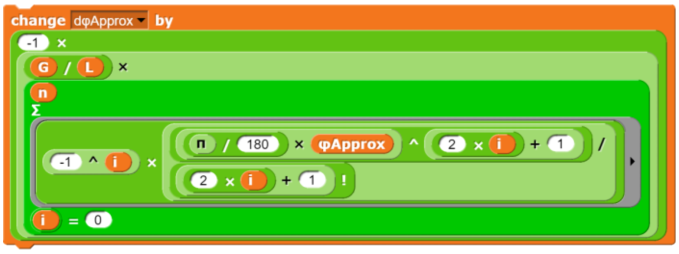
Instead of the linear approximation, we choose the Taylor series of the sine, of which we use *n* sums as approximation. We specify *n* as slider variable.

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

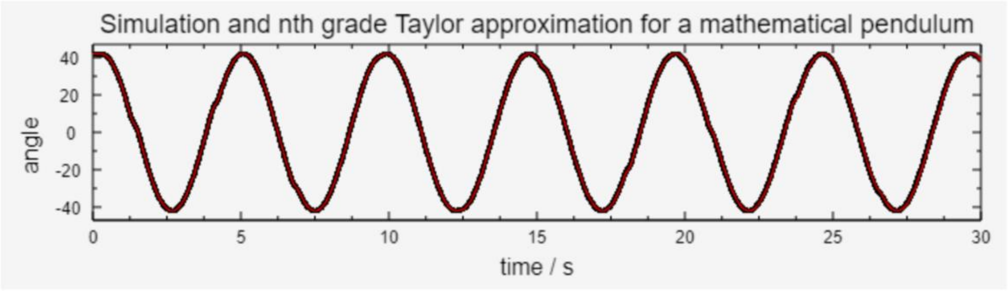
We can copy this directly into *SciSnap!*:



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

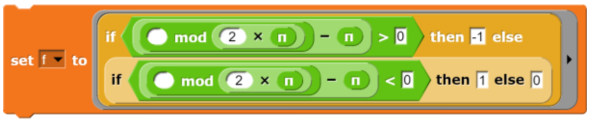
One way to Fourier represent a 2π -periodic function $f(x)$ is

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

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

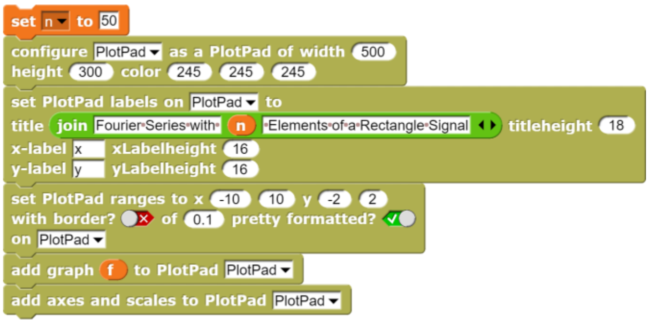
We want to determine the coefficients for the series expansion of a rectangular signal purely numerically and test how the length of the series expansion affects the accuracy of the results.

First of all we need a square wave signal with period 2π. A function that returns this would be e.g. $f(x) = \begin{cases} -1 & if\ (x\ mod\ 2\pi - \pi) > 0 \\ 1 & if\ (x\ mod\ 2\pi - \pi) < 0 \\ 0 & otherwise \end{cases}$

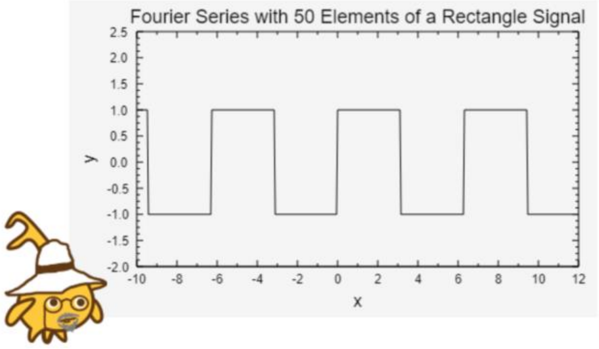


```
set f to if ((( ) mod (2 × π)) − (π)) > 0 then -1 else
         if ((( ) mod (2 × π)) − (π)) < 0 then 1 else 0
```

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

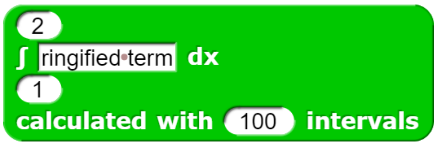


```
set n to 50
configure PlotPad as a PlotPad of width 500
height 300 color 245 245 245
set PlotPad labels on PlotPad to
title (join Fourier Series with (n) Elements of a Rectangle Signal) titleheight 18
x-label x xLabelheight 16
y-label y yLabelheight 16
set PlotPad ranges to x -10 10 y -2 2
with border?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



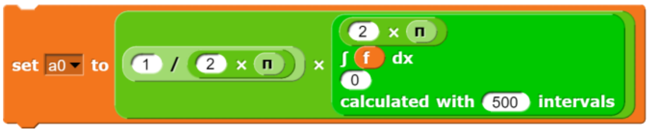
True so. 😉

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



```
2
∫ ringified term dx
1
calculated with 100 intervals
```

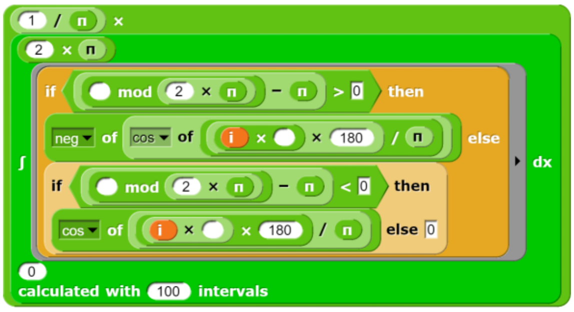
So, let's try it for $a_0 = \frac{1}{2\pi} \cdot \int_0^{2\pi} f(x) \cdot dx$. Since we already have the "ringified term" for the function *f*, we can simply write off:



```
set a0 to (1 / (2 × π)) × (2 × π
                            ∫ (f) dx
                            0
                            calculated with 500 intervals)
```

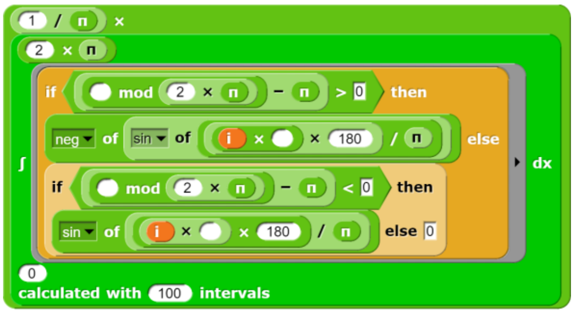
For the other coefficients we have to complete the trigonometric terms and obtain for

a_k:



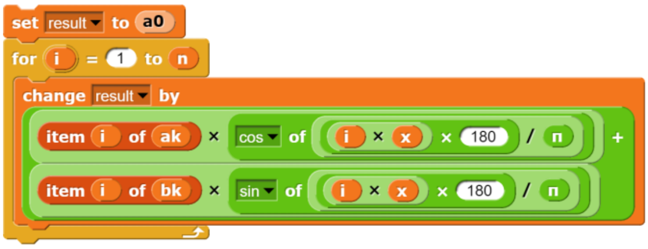
```
(1 / π) ×
2 × π
∫ if ((( ) mod (2 × π)) − (π)) > 0 then
      neg of (cos of (((i) × ( )) × 180 / (π)))
  else
      if ((( ) mod (2 × π)) − (π)) < 0 then
          cos of (((i) × ( )) × 180 / (π))
      else 0
  dx
0
calculated with 100 intervals
```

and b_k:



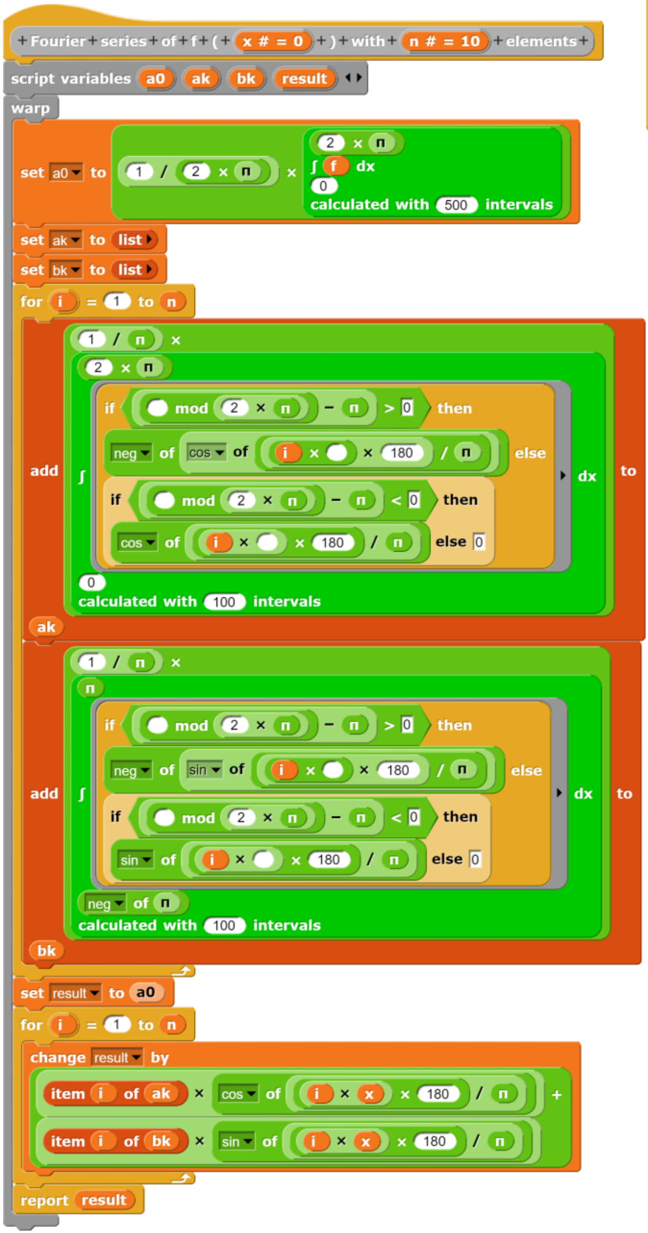
```
(1 / π) ×
2 × π
∫ if ((( ) mod (2 × π)) − (π)) > 0 then
      neg of (sin of (((i) × ( )) × 180 / (π)))
  else
      if ((( ) mod (2 × π)) − (π)) < 0 then
          sin of (((i) × ( )) × 180 / (π))
      else 0
  dx
0
calculated with 100 intervals
```

These terms now only have to be summed up:



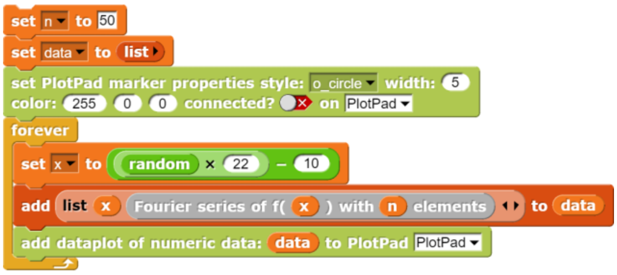
```
set result to a0
for i = 1 to n
    change result by
        (item i of ak) × cos of ((i × x × 180) / n) +
        (item i of bk) × sin of ((i × x × 180) / n)
```

This gives us the following script in total:

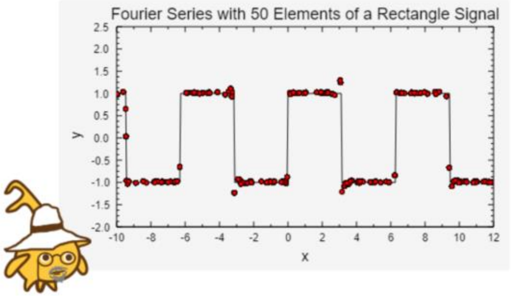


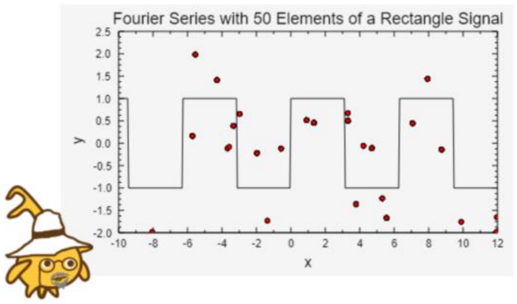
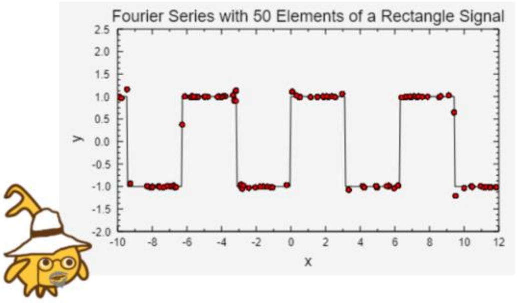
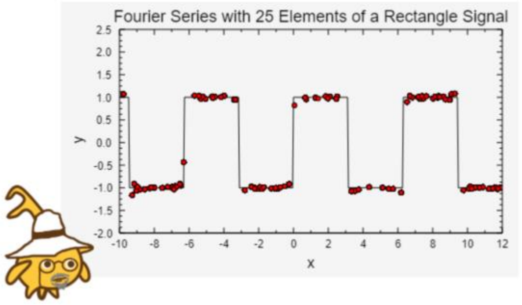
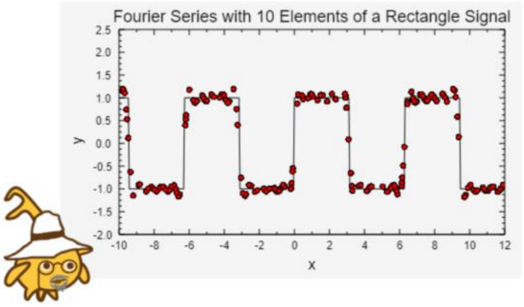
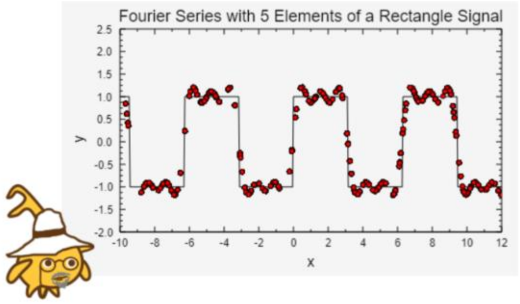
```
+Fourier+series+of+f+(+ x # = 0 +)+with+ n # = 10 +elements+
script variables a0 ak bk result ◂▸
warp
    set a0 to (1 / (2 × n)) ×   2 × n
                                ∫ f dx
                                0
                                calculated with 500 intervals
    set ak to list ▸
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

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.



```
set n to 50
set data to list ▸
set PlotPad marker properties style: o_circle width: 5
color: 255 0 0 connected? ☒ on PlotPad
forever
    set x to ((random × 22) − 10)
    add list x (Fourier series of f( x ) with n elements) ◂▸ to data
    add dataplot of numeric data: data to PlotPad PlotPad
```

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

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

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

Tasks:

1. a: Give function terms for a triangular signal, a signal made of peaks, ... Have the function graphs drawn.

   b: Calculate the Fourier series for the signals.

   c: Experiment as shown to find a reasonable tradeoff between accuracy and computation time requirements.

2. a: Create tables with the data for a triangle signal, a signal from peaks, ... using the function terms.

   b: Output the signals through the speaker.

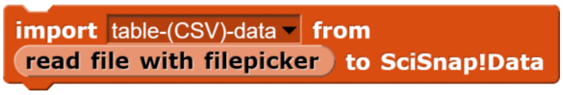
   c: Generate the corresponding tables using Fourier series of different quality and listen to them as sounds as well. Do you perceive any differences?

3. Find out about applications of Fourier series.

5.11 NY CitiBike Tripdata 1: Correlations

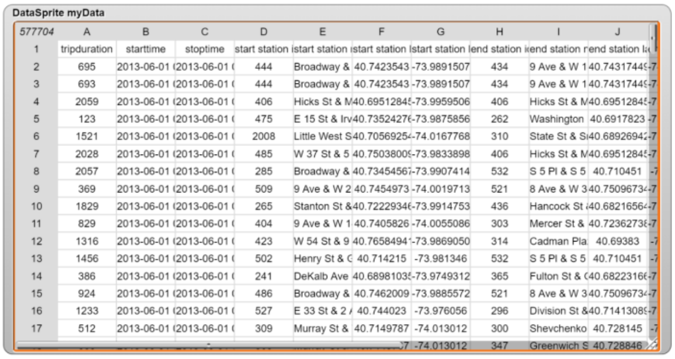
As an example of how to use the blocks, let's "dig" a bit into a larger, freely available dataset: the New York CitiBike rental data.
(NY citibike tripdata: https://www.citibikenyc. com/system-data)

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.



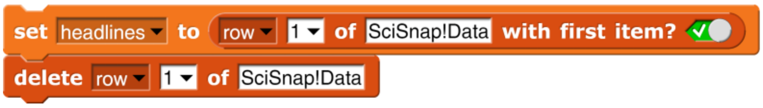
```
import table-(CSV)-data from read file with filepicker to SciSnap!Data
```

Their column headers we split off from the table.

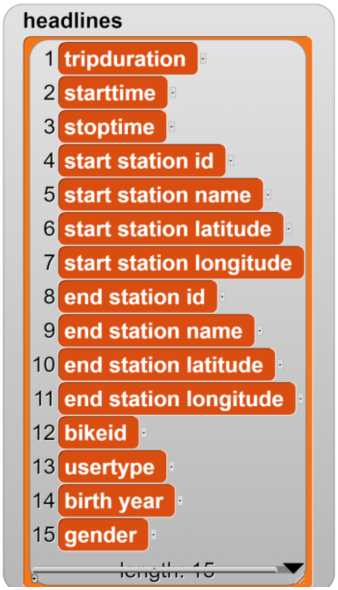


DataSprite myData

577704	A	B	C	D	E	F	G	H	I	J
1	tripduration	starttime	stoptime	start station	start station	start station	start station	end station	end station	end station la
2	695	2013-06-01 (	2013-06-01 (	444	Broadway &	40.7423543	-73.9891507	434	9 Ave & W 1	40.7431744
3	693	2013-06-01 (	2013-06-01 (	444	Broadway &	40.7423543	-73.9891507	434	9 Ave & W 1	40.7431744
4	2059	2013-06-01 (	2013-06-01 (	406	Hicks St & M	40.6951284	-73.9959506	406	Hicks St & M	40.6951284
5	123	2013-06-01 (	2013-06-01 (	475	E 15 St & Irv	40.7352427	-73.9875856	262	Washington	40.6917823
6	1521	2013-06-01 (	2013-06-01 (	2008	Little West S	40.7056925	-74.0167768	310	State St & S	40.6892694
7	2028	2013-06-01 (	2013-06-01 (	485	W 37 St & 5	40.7503800	-73.9833898	406	Hicks St & M	40.6951284
8	2057	2013-06-01 (	2013-06-01 (	285	Broadway &	40.7345456	-73.9907414	532	S 5 Pl & S 5	40.710451
9	369	2013-06-01 (	2013-06-01 (	509	9 Ave & W 2	40.7454973	-74.0019713	521	8 Ave & W 3	40.7509673
10	1829	2013-06-01 (	2013-06-01 (	265	Stanton St &	40.7222934	-73.9914753	436	Hancock St	40.6821656
11	829	2013-06-01 (	2013-06-01 (	404	9 Ave & W 1	40.7405826	-74.0055086	303	Mercer St &	40.7236273
12	1316	2013-06-01 (	2013-06-01 (	423	W 54 St & 9	40.7658494	-73.9869050	314	Cadman Pla	40.69383
13	1456	2013-06-01 (	2013-06-01 (	502	Henry St & C	40.714215	-73.981346	532	S 5 Pl & S 5	40.710451
14	386	2013-06-01 (	2013-06-01 (	241	DeKalb Ave	40.6898103	-73.9749312	365	Fulton St & (	40.6822316
15	924	2013-06-01 (	2013-06-01 (	486	Broadway &	40.7462009	-73.9885572	521	8 Ave & W 3	40.7509673
16	1233	2013-06-01 (	2013-06-01 (	527	E 33 St & 2	40.744023	-73.976056	296	Division St &	40.7141308
17	512	2013-06-01 (	2013-06-01 (	309	Murray St &	40.7149787	-74.013012	300	Shevchenko	40.728145
							-74.013012	347	Greenwich S	40.728846



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

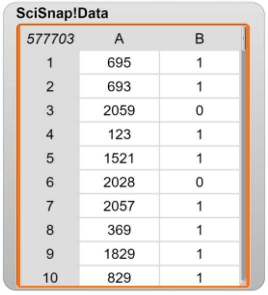


headlines

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

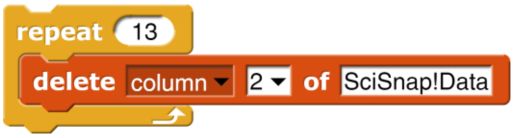
length: 15

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.



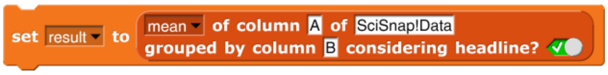
SciSnap!Data

577703	A	B
1	695	1
2	693	1
3	2059	0
4	123	1
5	1521	1
6	2028	0
7	2057	1
8	369	1
9	1829	1
10	829	1

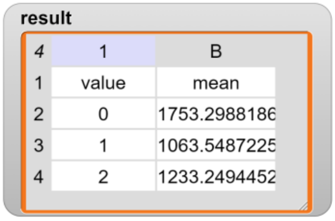


```
repeat 13
    delete column 2 of SciSnap!Data
```

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



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

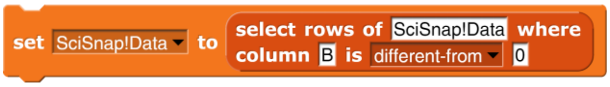


result

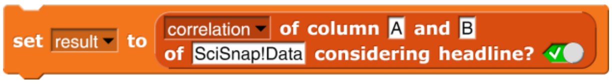
4	1	B
1	value	mean
2	0	1753.2988186
3	1	1063.5487225
4	2	1233.2494452

That's what we thought! 😉

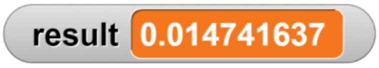
And what about the correlation? First, we throw out the data with the "unknown" gender. There are still about 340,000 data records left. For these, we calculate the correlation coefficient between column 1 and 2 - and get the result shown opposite.



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



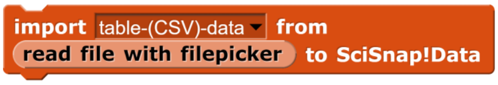
```
set result to correlation of column A and B of SciSnap!Data considering headline? ✓
```



result 0.014741637

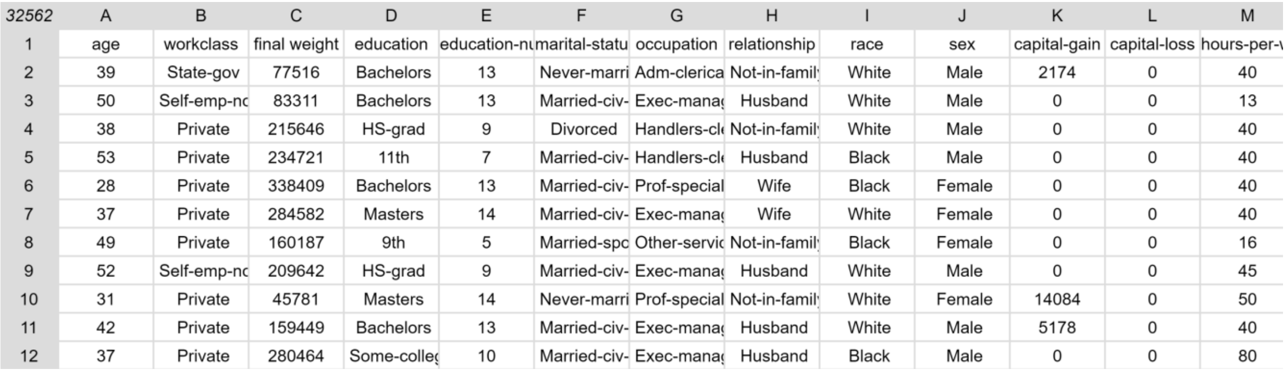
And what does this number want to tell us now??? We don't know - but we can read up and learn! 😉

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



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

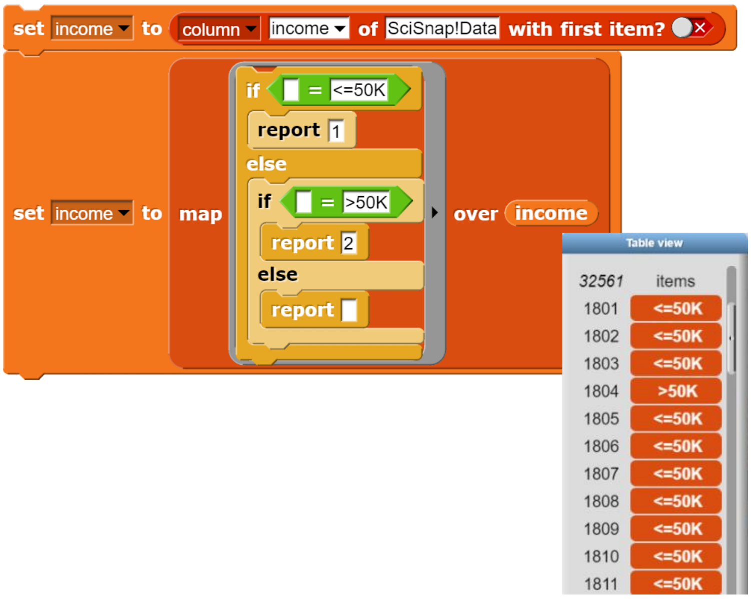
We want to dig into some data and therefore download the *Census Income Dataset* from the net.[20] The corresponding CSV file can be loaded from the file directory in *SciSnap!Data* and displayed immediately. It contains 32562 records. Double-clicking or right-clicking on it and selecting "*open in dialog...*" shows all columns.



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

Which correlations could be found in this?

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

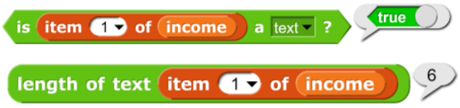


```
set income to column income of SciSnap!Data with first item? ✕
set income to map (
    if ( □ = <=50K )
        report 1
    else
        if ( □ = >50K )
            report 2
        else
            report □
) over income
```

Table view

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

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.

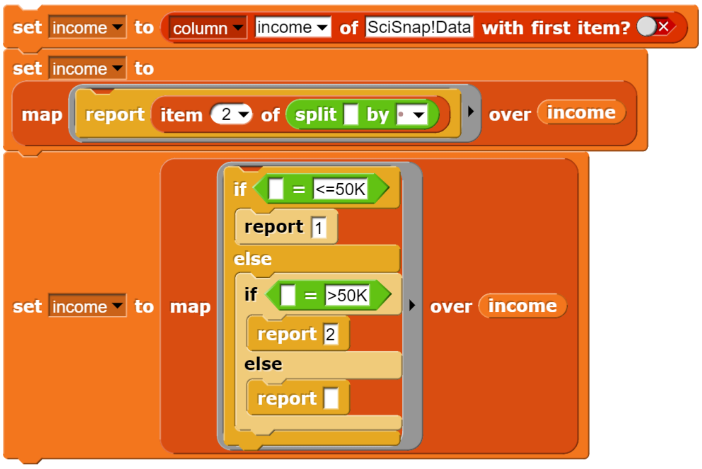


```
is item 1 of income a text ?          → true
length of text item 1 of income       → 6
```

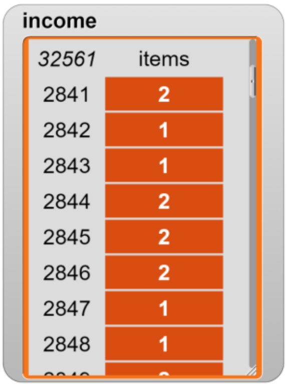
What's going on? We look at the first element of *income* and check if it is a string. It is, but it is longer than we thought:

[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.



```
set income to column income of SciSnap!Data with first item? ☒
set income to map (report item 2 of split [] by [ ]) over income
set income to map (
    if [] = <=50K
        report 1
    else
        if [] = >50K
            report 2
        else
            report []
) over income
```

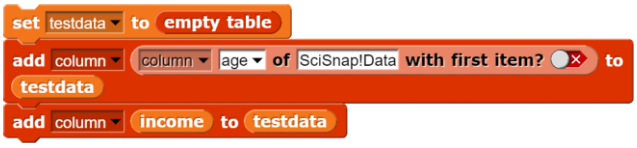


income

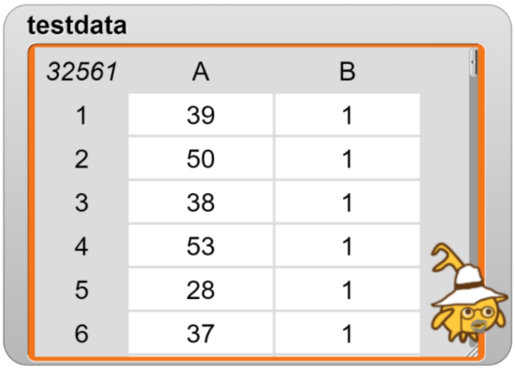
32561	items
2841	2
2842	1
2843	1
2844	2
2845	2
2846	2
2847	1
2848	1

What does this income now depend on?

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



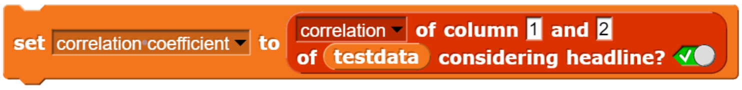
```
set testdata to empty table
add column (column age of SciSnap!Data with first item? ☒) to testdata
add column income to testdata
```



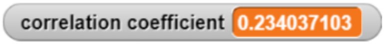
testdata

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

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



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



correlation coefficient 0.234037103

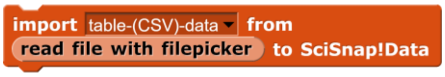
And what does that mean?

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.



```
import table-(CSV)-data from
read file with filepicker to SciSnap!Data
```

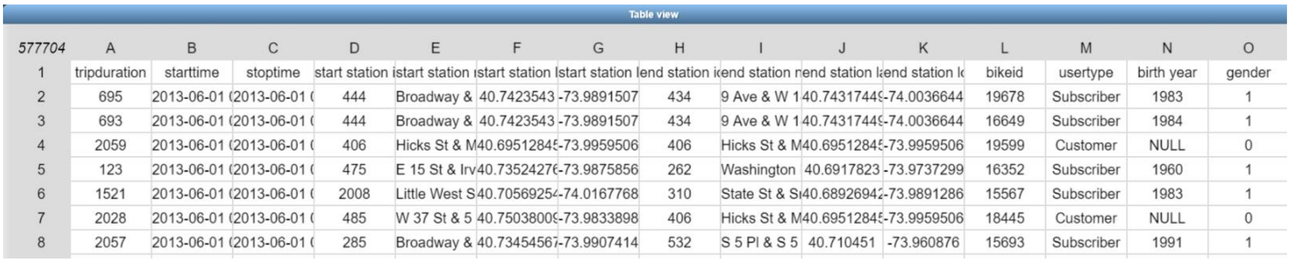
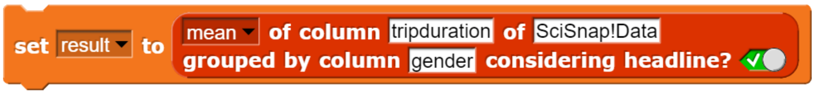


Table view

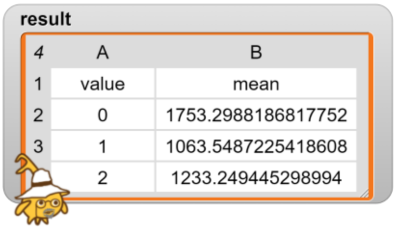
577704	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	tripduration	starttime	stoptime	start station i	start station ı	start station l	start station l	end station i	end station n	end station l	end station l	bikeid	usertype	birth year	gender
2	695	2013-06-01 (	2013-06-01 (	444	Broadway &	40.7423543	-73.9891507	434	9 Ave & W 1	40.7431744	-74.0036644	19678	Subscriber	1983	1
3	693	2013-06-01 (	2013-06-01 (	444	Broadway &	40.7423543	-73.9891507	434	9 Ave & W 1	40.7431744	-74.0036644	16649	Subscriber	1984	1
4	2059	2013-06-01 (	2013-06-01 (	406	Hicks St & M	40.6951284	-73.9959506	406	Hicks St & M	40.6951284	-73.9959506	19599	Customer	NULL	0
5	123	2013-06-01 (	2013-06-01 (	475	E 15 St & Irv	40.7352427	-73.9875856	262	Washington	40.6917823	-73.9737299	16352	Subscriber	1960	1
6	1521	2013-06-01 (	2013-06-01 (	2008	Little West S	40.7056925	-74.0167768	310	State St & S	40.6892694	-73.9891286	15567	Subscriber	1983	1
7	2028	2013-06-01 (	2013-06-01 (	485	W 37 St & 5	40.7503800	-73.9833898	406	Hicks St & M	40.6951284	-73.9959506	18445	Customer	NULL	0
8	2057	2013-06-01 (	2013-06-01 (	285	Broadway &	40.7345456	-73.9907414	532	S 5 Pl & S 5	40.710451	-73.960876	15693	Subscriber	1991	1

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

The average duration of borrowing, in terms of gender:



```
set result to mean of column tripduration of SciSnap!Data
grouped by column gender considering headline? ✓
```

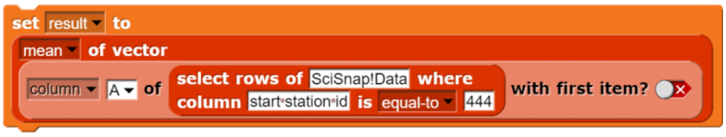


result

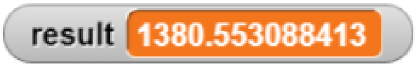
4	A	B
1	value	mean
2	0	1753.2988186817752
3	1	1063.5487225418608
	2	1233.249445298994

That's what we thought!

Let's see if they are lazier on Broadway:

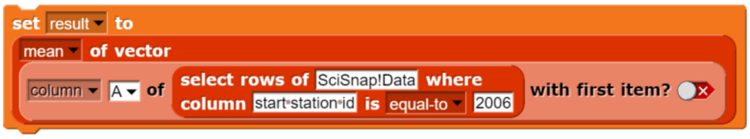


```
set result to
mean of vector
column A of select rows of SciSnap!Data where
column start station id is equal-to 444 with first item? ✗
```

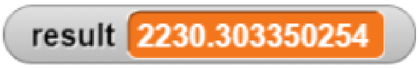


result 1380.553088413

I see. Probably Central Park is even worse!



```
set result to
mean of vector
column A of select rows of SciSnap!Data where
column start station id is equal-to 2006 with first item? ✗
```



result 2230.303350254

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

Tasks:

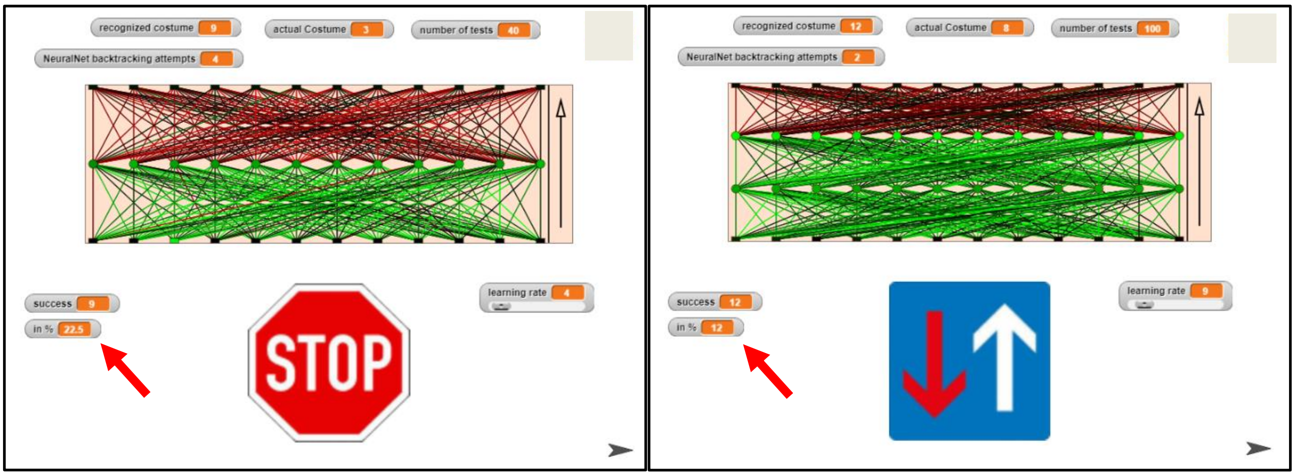
1. But maybe only the women at Central Park ride their bikes more. Check it out.
2. There is not only one rental station at Central Park. Determine appropriate averages for the entire area.
3. Are there actually rental data for other parts of the city? Do a search and compare the results with Manhattan.
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.



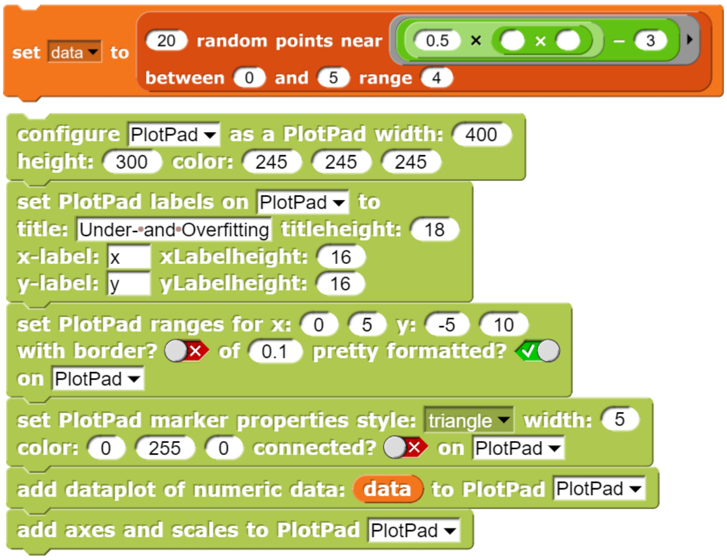
Re 2: If the training data is well reproduced by the function, it does not mean that this is also true for other data. It depends very much on the kind of function that is generated. As application we choose the example *polynomial interpolation*.

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 $0{,}5 * x^2 - 3$.

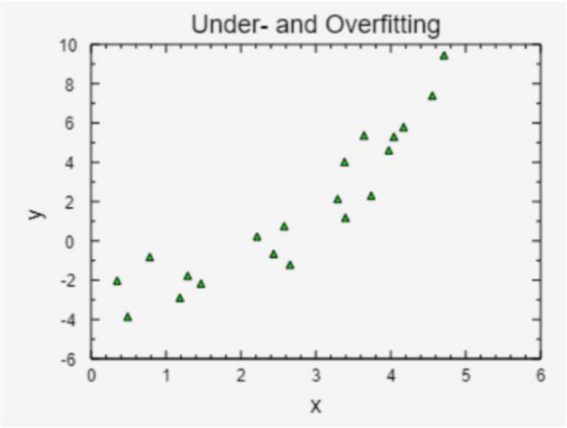
We configure a second sprite next to *Hilberto* named *PlotPad* as *PlotPad* and plot the data on it.

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

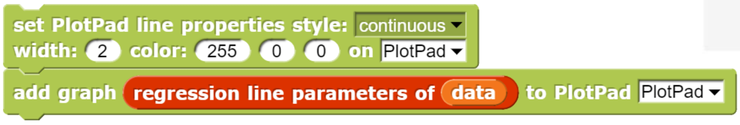


```
set data ▾ to 20 random points near ((0.5 × ( ) × ( )) − 3) ▸
              between 0 and 5 range 4
configure PlotPad ▾ as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on PlotPad ▾ to
title: Under-and-Overfitting titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16
set PlotPad ranges for x: 0 5 y: -5 10
with border? ☒ of 0.1 pretty formatted? ☑
on PlotPad ▾
set PlotPad marker properties style: triangle ▾ width: 5
color: 0 255 0 connected? ☒ on PlotPad ▾
add dataplot of numeric data: data to PlotPad PlotPad ▾
add axes and scales to PlotPad PlotPad ▾
```

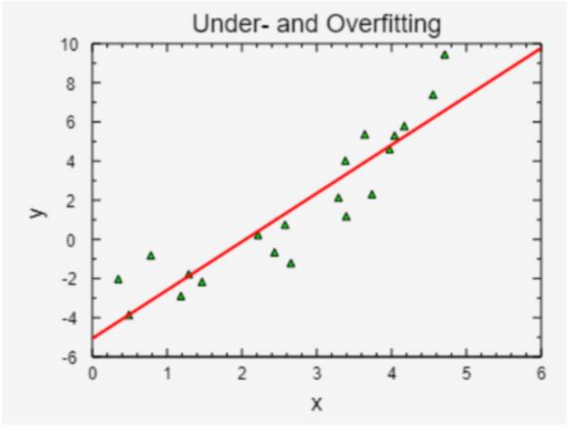
This is already enough to show the data.



We first try the interpolation with a regression line.



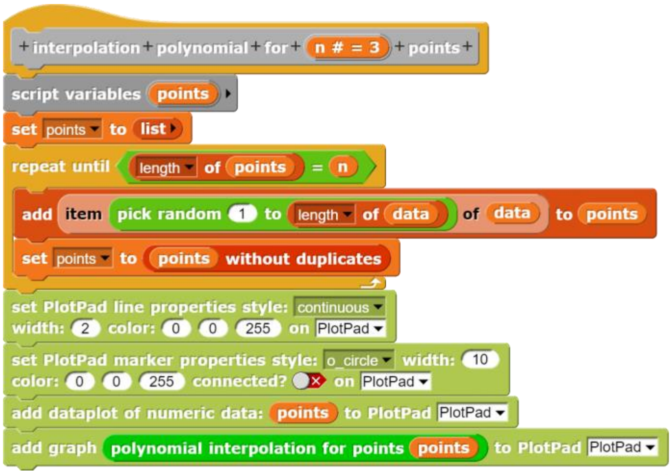
```
set PlotPad line properties style: continuous
width: 2 color: 255 0 0 on PlotPad
add graph (regression line parameters of data) to PlotPad PlotPad
```



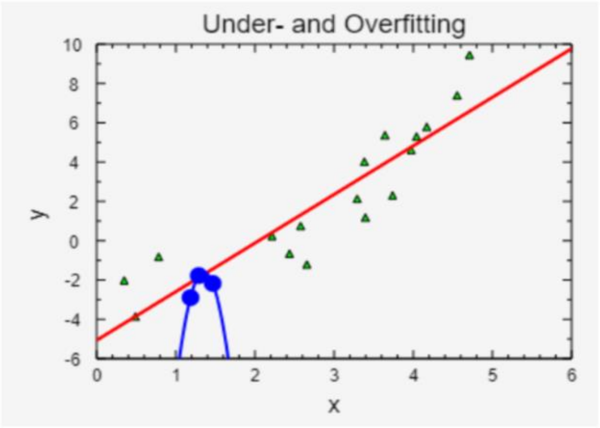
This actually looks quite nice, but it doesn't properly fit on the sides.

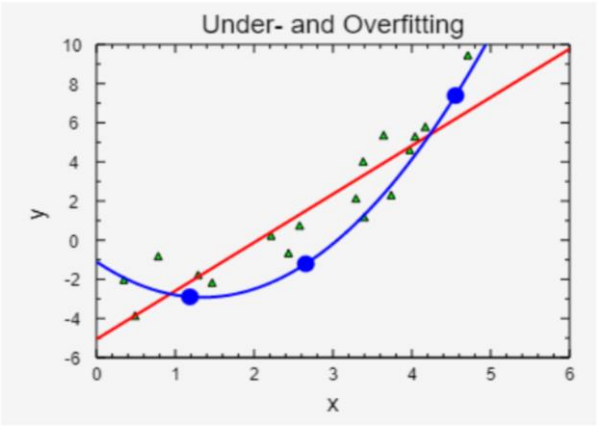
So, we try it with a polynomial interpolation.

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

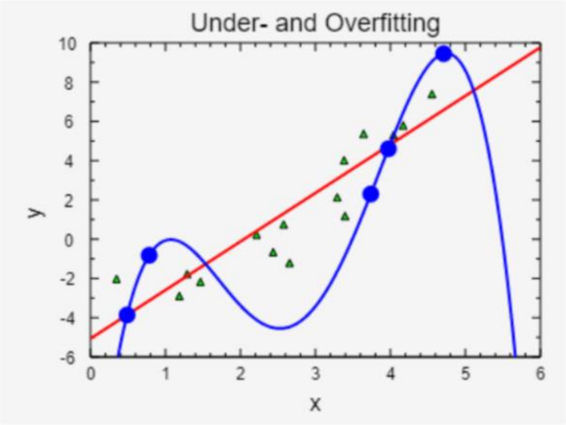


```
+interpolation+polynomial+for+(n # = 3)+points+
script variables (points)
set points to (list)
repeat until ((length of points) = n)
    add item (pick random 1 to (length of data)) of data to points
    set points to (points without duplicates)
set PlotPad line properties style: continuous
width: 2 color: 0 0 255 on PlotPad
set PlotPad marker properties style: o_circle width: 10
color: 0 0 255 connected? on PlotPad
add dataplot of numeric data: points to PlotPad PlotPad
add graph (polynomial interpolation for points points) to PlotPad PlotPad
```

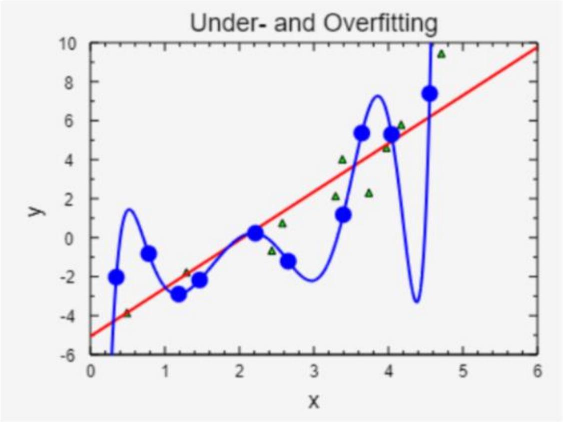


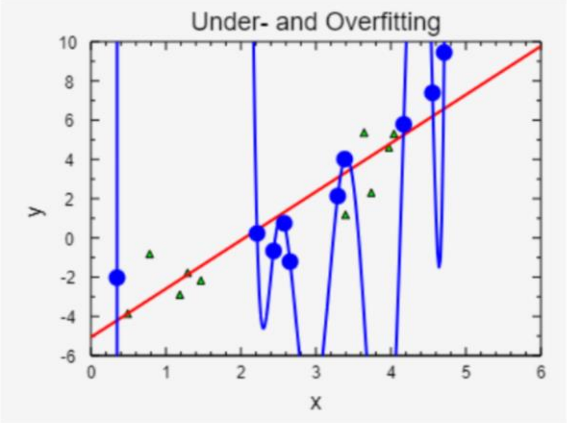


Now we're getting brave! Instead of three points, we choose 5. After all, we want to do a good job! That works halfway in the middle, and then - oops!

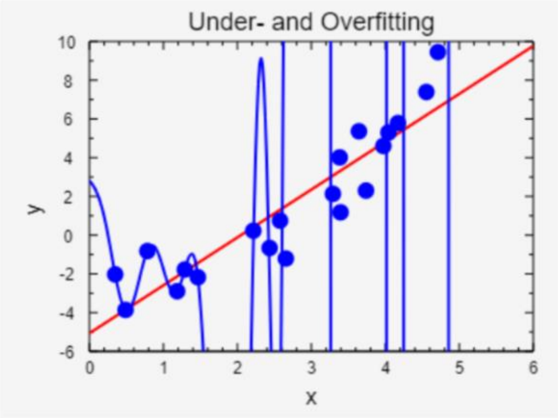


Maybe we just have to use more points. Let's try 10. The polynomials run through more points, but they "take off" at the edges.





Well, then with all the points!



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

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

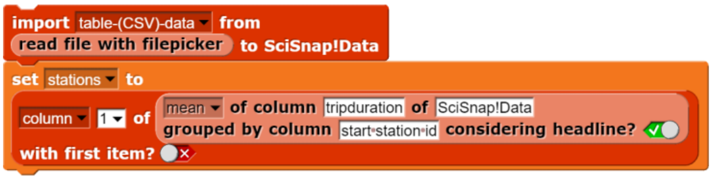
Tasks:

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

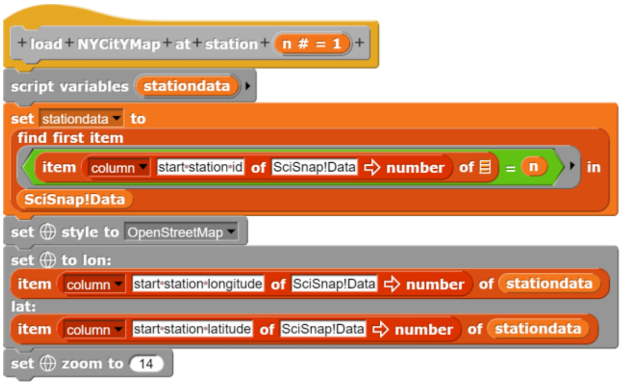
5.15 New York Citibike Tripdata 3: World Map Library

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.

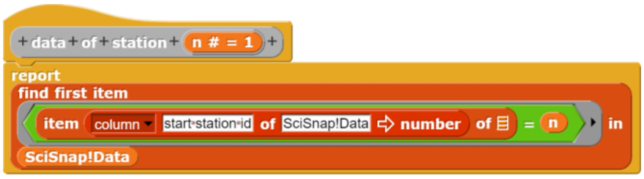


```
import table-(CSV)-data from read file with filepicker to SciSnap!Data
set stations to
    column 1 of (mean of column tripduration of SciSnap!Data
                 grouped by column start-station-id considering headline? ✓)
    with first item? ✗
```



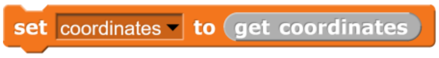
```
+load+NYCityMap+at+station+ (n # = 1) +
script variables stationdata
set stationdata to
    find first item ((item (column start-station-id of SciSnap!Data ⇨ number) of ☰) = n) in
    SciSnap!Data
set 🌐 style to OpenStreetMap
set 🌐 to lon:
    item (column start-station-longitude of SciSnap!Data ⇨ number) of stationdata
lat:
    item (column start-station-latitude of SciSnap!Data ⇨ number) of stationdata
set 🌐 zoom to 14
```

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

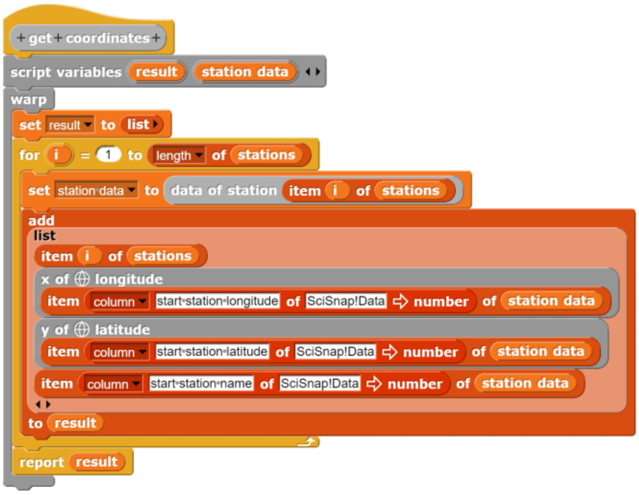


```
+data+of+station+ (n # = 1) +
report
    find first item ((item (column start-station-id of SciSnap!Data ⇨ number) of ☰) = n) in
    SciSnap!Data
```

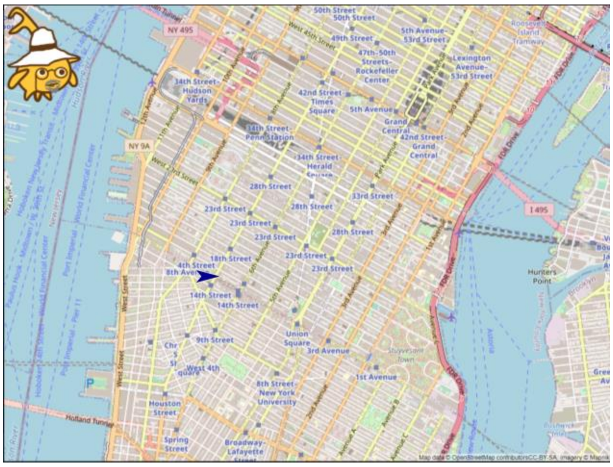
... and thus set up the list of coordinates of the stations [21].

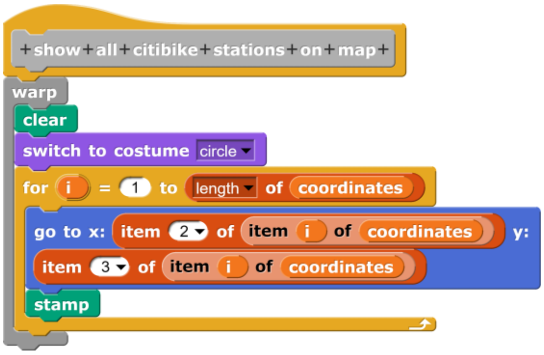


```
set coordinates to get coordinates
```



```
+get+coordinates+
script variables result station data
warp
    set result to list
    for i = 1 to length of stations
        set station data to data of station item i of stations
        add list
                item i of stations
                x of 🌐 longitude
                    item (column start-station-longitude of SciSnap!Data ⇨ number) of station data
                y of 🌐 latitude
                    item (column start-station-latitude of SciSnap!Data ⇨ number) of station data
                item (column start-station-name of SciSnap!Data ⇨ number) of station data
            to result
report result
```



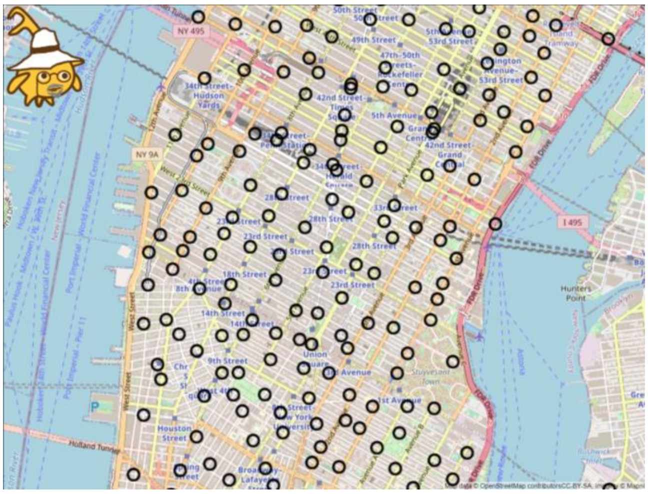


```
+show+all+citibike+stations+on+map+
warp
    clear
    switch to costume circle
    for i = 1 to length of coordinates
        go to x: item 2 of item i of coordinates y:
                 item 3 of item i of coordinates
        stamp
```

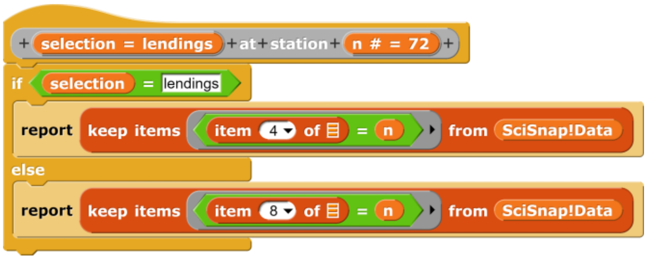
With this data we can send *Hilberto* to the individual positions and ask him to leave circles there with the stamp block, for example.

[21] This will take some time!

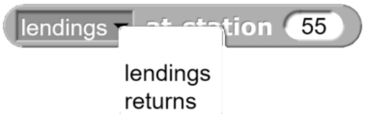
At least in midtown Manhattan, I don't think we have to worry about finding a borrowing station!



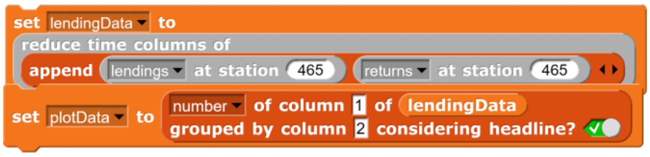
Let's take a closer look at the rental station Broadway - corner 41 Street (No. 465). To do this, we pull all records from the total list that start or end at this station. That's 5054 operations this month. Times are entered in this list along with the date. We can throw this out (*split with " "*) 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.



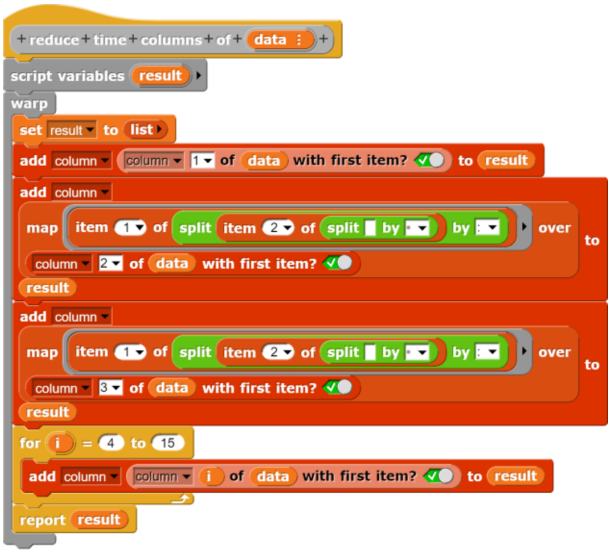
```
+ selection = lendings + at + station + n # = 72 +
if selection = lendings
  report keep items (item 4 of ⌸ = n) from SciSnap!Data
else
  report keep items (item 8 of ⌸ = n) from SciSnap!Data
```



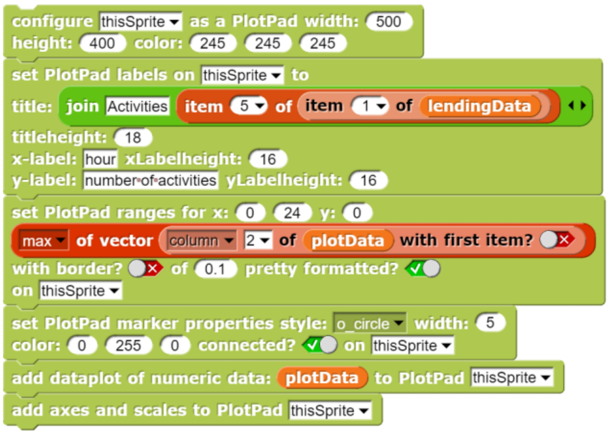
```
lendings ▾ at station 55
lendings
returns
```



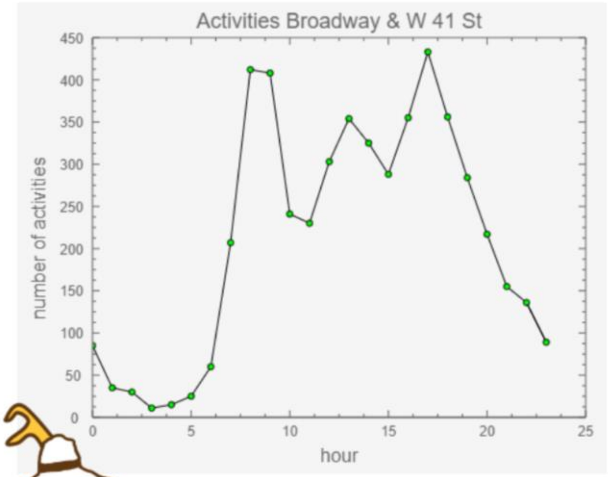
```
set lendingData ▾ to
  reduce time columns of
    append (lendings ▾ at station 465) (returns ▾ at station 465) ◂▸
set plotData ▾ to number ▾ of column 1 of lendingData
  grouped by column 2 considering headline? ✓
```



```
+ reduce + time + columns + of + data ⁝ +
script variables result ▸
warp
  set result ▾ to list ▸
  add column ▾ (column ▾ 1 ▾ of data with first item? ✓) to result
  add column ▾ (map (item 1 ▾ of split (item 2 ▾ of split [ ] by [·] ▾) by [:] ▾) ▸ over
      (column ▾ 2 ▾ of data with first item? ✓)) to
    result
  add column ▾ (map (item 1 ▾ of split (item 2 ▾ of split [ ] by [·] ▾) by [:] ▾) ▸ over
      (column ▾ 3 ▾ of data with first item? ✓)) to
    result
  for i = 4 to 15
    add column ▾ (column ▾ i of data with first item? ✓) to result
  report result
```

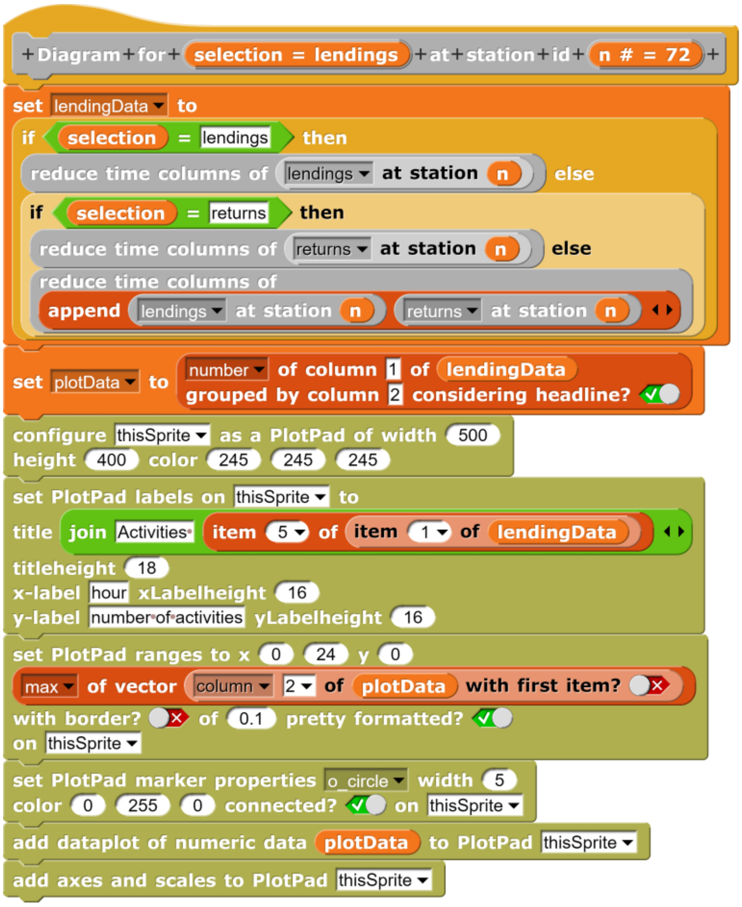


```
configure thisSprite ▾ as a PlotPad width: 500
  height: 400 color: 245 245 245
set PlotPad labels on thisSprite ▾ to
  title: join Activities (item 5 ▾ of item 1 ▾ of lendingData) ◂▸
  titleheight: 18
  x-label: hour xLabelheight: 16
  y-label: number·of·activities yLabelheight: 16
set PlotPad ranges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



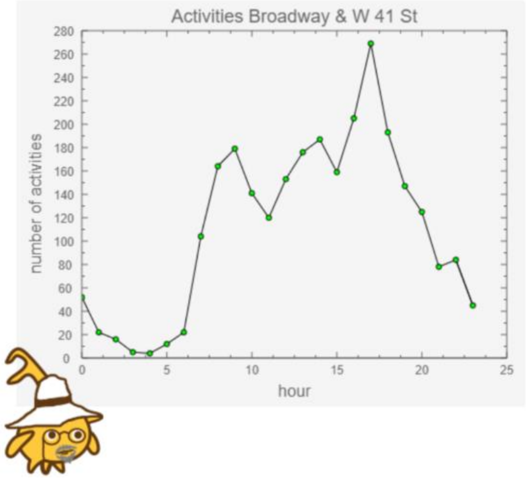
And we can represent this graphically as usual.

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

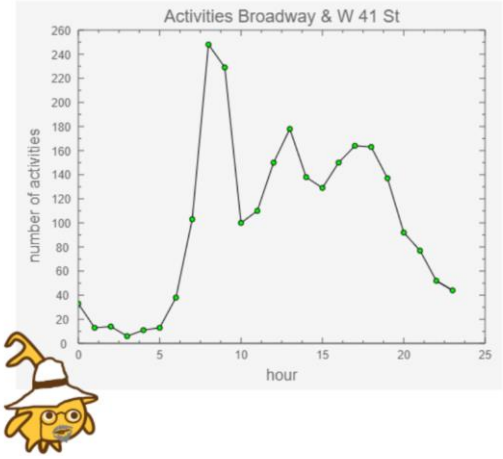


```
+Diagram+for+ selection = lendings +at+station+id+ n # = 72 +
set lendingData to
  if selection = lendings then
    reduce time columns of lendings at station n else
  if selection = returns then
    reduce time columns of returns at station n else
    reduce time columns of
    append lendings at station n returns at station n
set plotData to number of column 1 of lendingData
    grouped by column 2 considering headline? ✓
configure thisSprite as a PlotPad of width 500
height 400 color 245 245 245
set PlotPad labels on thisSprite to
title join Activities item 5 of item 1 of lendingData
titleheight 18
x-label hour xLabelheight 16
y-label number of activities yLabelheight 16
set PlotPad ranges to x 0 24 y 0
  max of vector column 2 of plotData with first item? ☒
with border? ☒ of 0.1 pretty formatted? ✓
on thisSprite
set PlotPad marker properties o_circle width 5
color 0 255 0 connected? ✓ on thisSprite
add dataplot of numeric data plotData to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```

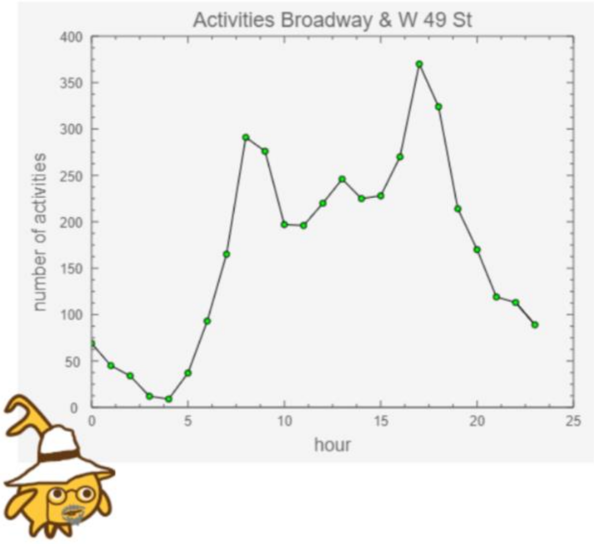
Lending:

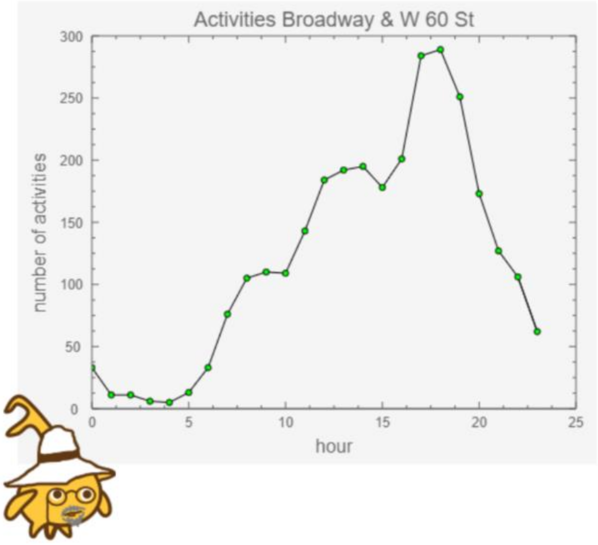


Returns:



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.

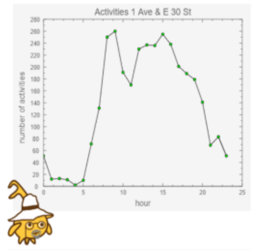
So, what can our programs learn from this data?

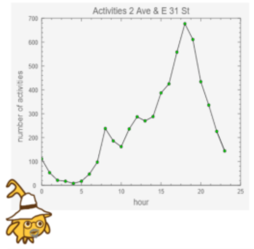
- *For example, we could predict from the usual departures and arrivals as well as the actual stock whether enough bikes will be returned to a station in time or whether it would be better to transport some there.*
- *We could determine from mean path lengths which batteries are being used for eBikes.*
- *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.*

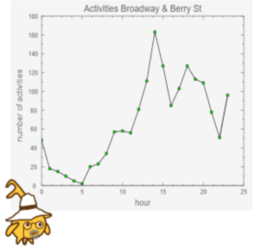
etc.

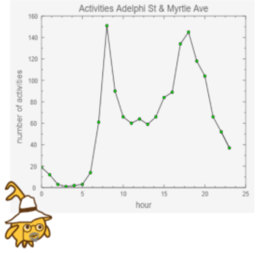
Tasks:

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.







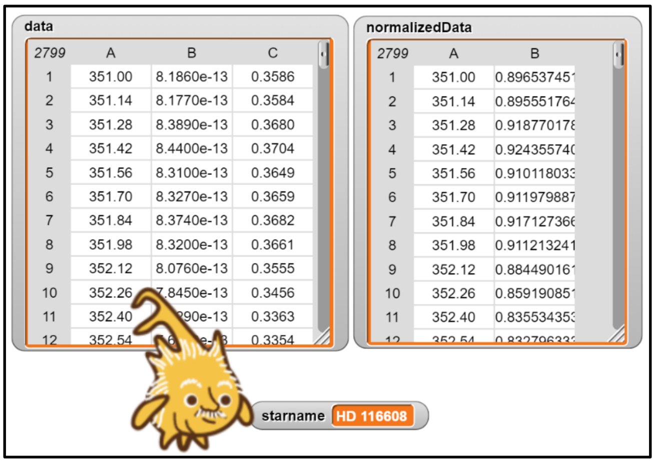


5.16 Star spectra [UniGOE]

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*.



data

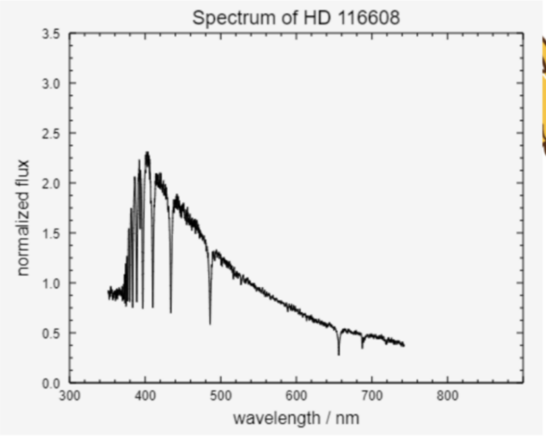
2799	A	B	C
1	351.00	8.1860e-13	0.3586
2	351.14	8.1770e-13	0.3584
3	351.28	8.3890e-13	0.3680
4	351.42	8.4400e-13	0.3704
5	351.56	8.3100e-13	0.3649
6	351.70	8.3270e-13	0.3659
7	351.84	8.3740e-13	0.3682
8	351.98	8.3200e-13	0.3661
9	352.12	8.0760e-13	0.3555
10	352.26	7.8450e-13	0.3456
11	352.40	[illegible]290e-13	0.3363
12	352.54	[illegible]	0.3354

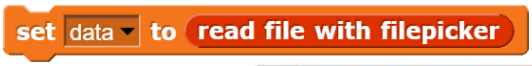
normalizedData

2799	A	B
1	351.00	0.89653745
2	351.14	0.89555176
3	351.28	0.91877017
4	351.42	0.92435574
5	351.56	0.91011803
6	351.70	0.91197988
7	351.84	0.91712736
8	351.98	0.91121324
9	352.12	0.88449016
10	352.26	0.85919085
11	352.40	0.83553435
12	352.54	0.83279633

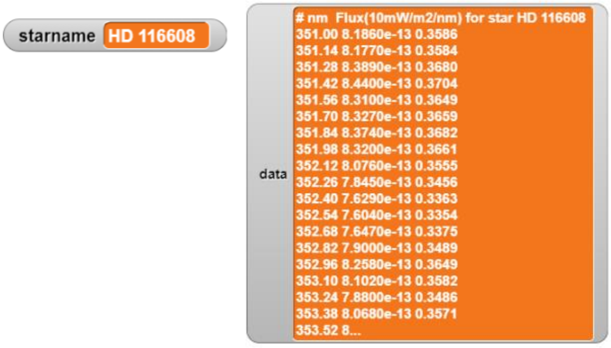
starname HD 116608

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





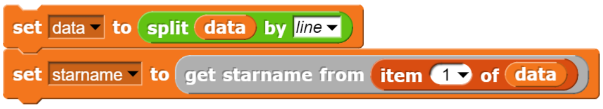
```
set data to read file with filepicker
```



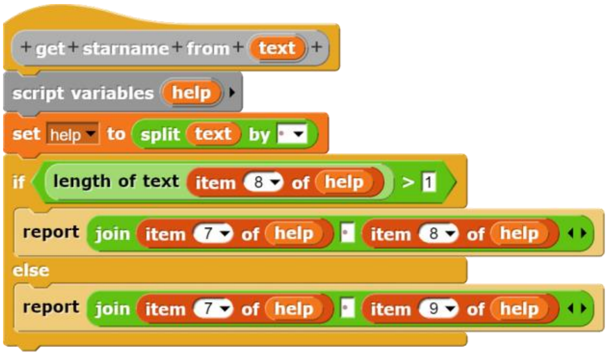
starname HD 116608

data

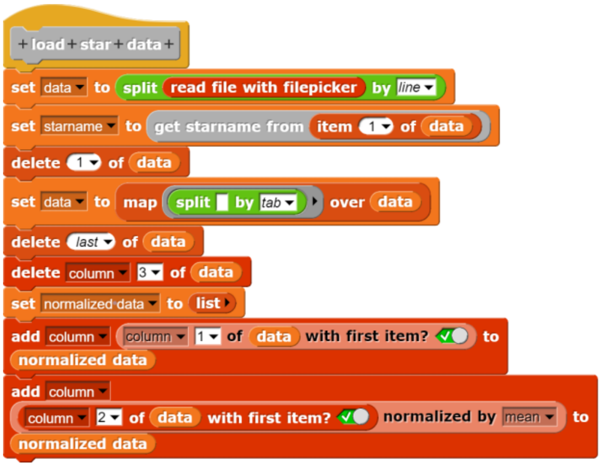
```
# nm  Flux(10mW/m2/nm) for star HD 116608
351.00 8.1860e-13 0.3586
351.14 8.1770e-13 0.3584
351.28 8.3890e-13 0.3680
351.42 8.4400e-13 0.3704
351.56 8.3100e-13 0.3649
351.70 8.3270e-13 0.3659
351.84 8.3740e-13 0.3682
351.98 8.3200e-13 0.3661
352.12 8.0760e-13 0.3555
352.26 7.8450e-13 0.3456
352.40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



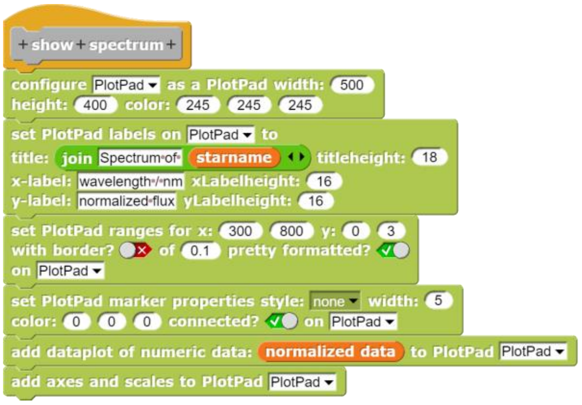
```
set data to split data by line
set starname to get starname from item 1 of data
```



```
+get+starname+from+ text +
script variables help
set help to split text by ·
if length of text item 8 of help > 1
    report join item 7 of help · item 8 of help
else
    report join item 7 of help · item 9 of help
```

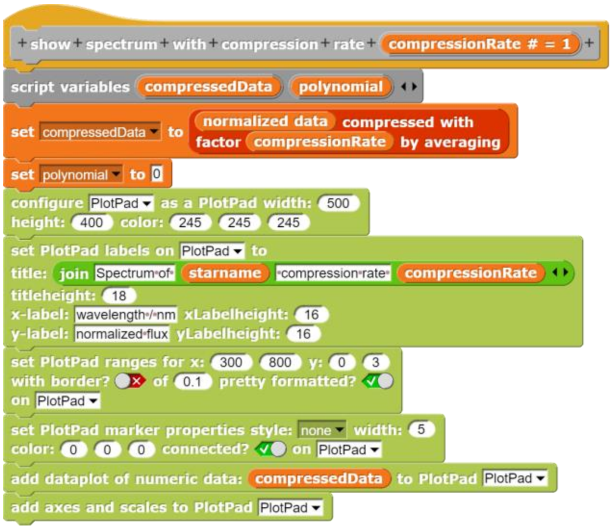


```
+load+star+data+
set data to split read file with filepicker by line
set starname to get starname from item 1 of data
delete 1 of data
set data to map split [ ] by tab over data
delete last of data
delete column 3 of data
set normalized data to list
add column column 1 of data with first item? ✓ to
normalized data
add column
column 2 of data with first item? ✓ normalized by mean to
normalized data
```

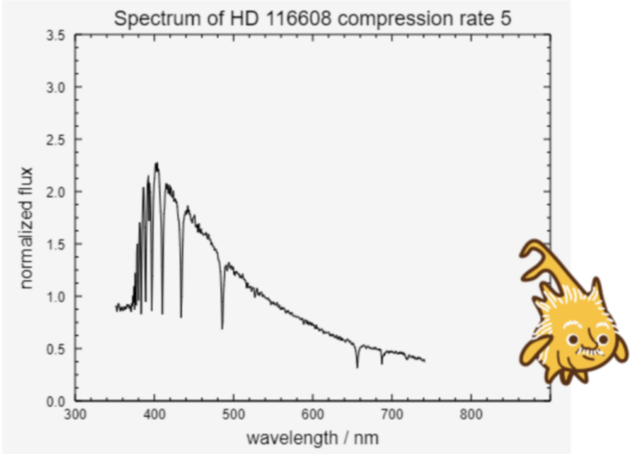


```
+show+spectrum+
configure PlotPad as a PlotPad width: 500
height: 400 color: 245 245 245
set PlotPad labels on PlotPad to
title: join Spectrum·of· starname titleheight: 18
x-label: wavelength·/·nm xLabelheight: 16
y-label: normalized·flux yLabelheight: 16
set PlotPad ranges for x: 300 800 y: 0 3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

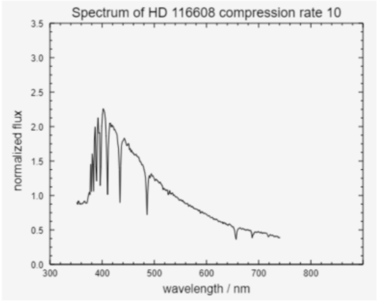
One can see well the sloping course with some striking absorption lines. But do we need all spectral data at all for this insight? Maybe it is enough to reduce the amount of data by averaging. We introduce a compression factor *compressionRate* and complete the script before the diagram generation.

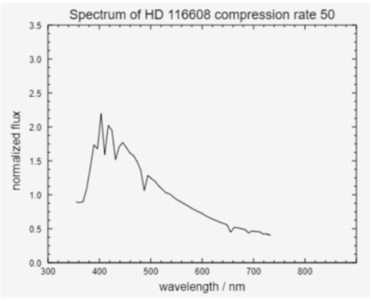


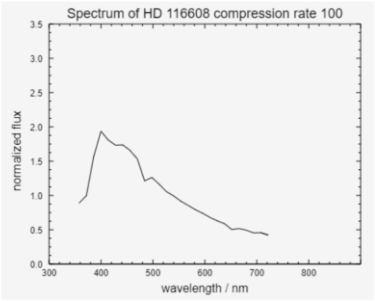
```
+show+spectrum+with+compression+rate+ compressionRate # = 1 +
script variables compressedData polynomial
set compressedData to normalized data compressed with
    factor compressionRate by averaging
set polynomial to 0
configure PlotPad as a PlotPad width: 500
height: 400 color: 245 245 245
set PlotPad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



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

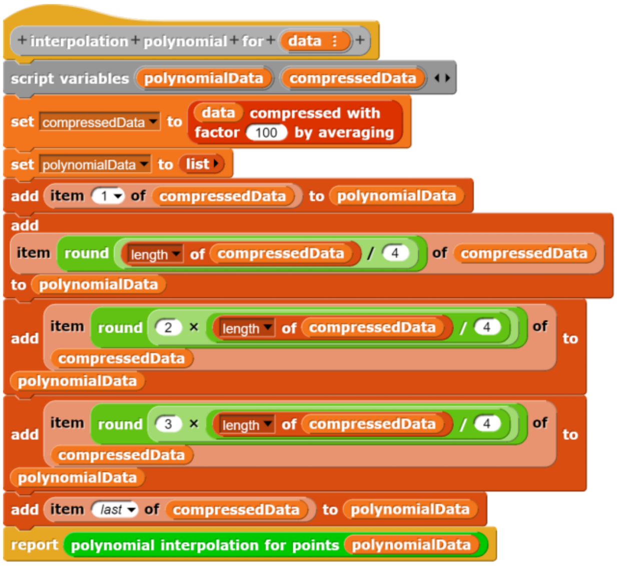




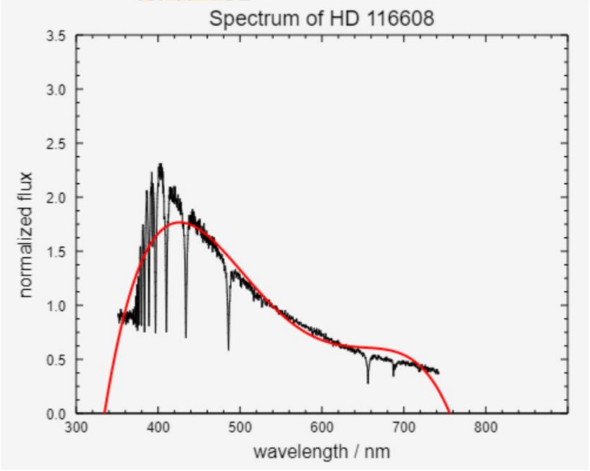


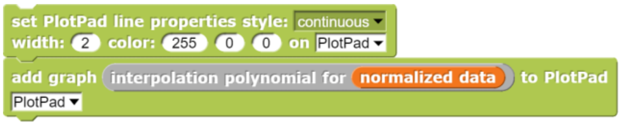
It can be seen that the temperature-dependent course of the spectrum is hardly changed. Only the absorption lines are lost.

Thus, the type of the spectrum should be described by an interpolation polynomial of e.g. 4th degree.



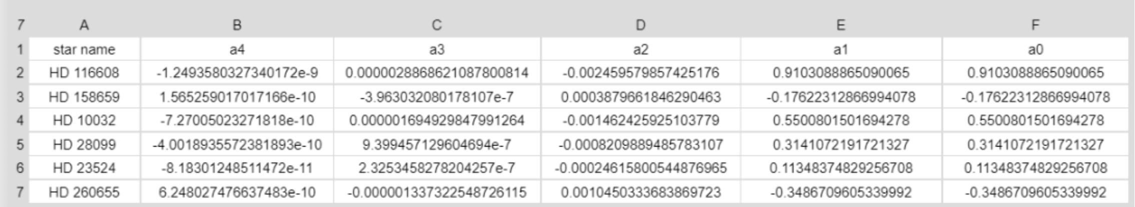
```
+interpolation+polynomial+for+ data : +
script variables polynomialData compressedData
set compressedData to data compressed with
    factor 100 by averaging
set polynomialData to list
add item 1 of compressedData to polynomialData
add
    item round length of compressedData / 4 of compressedData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



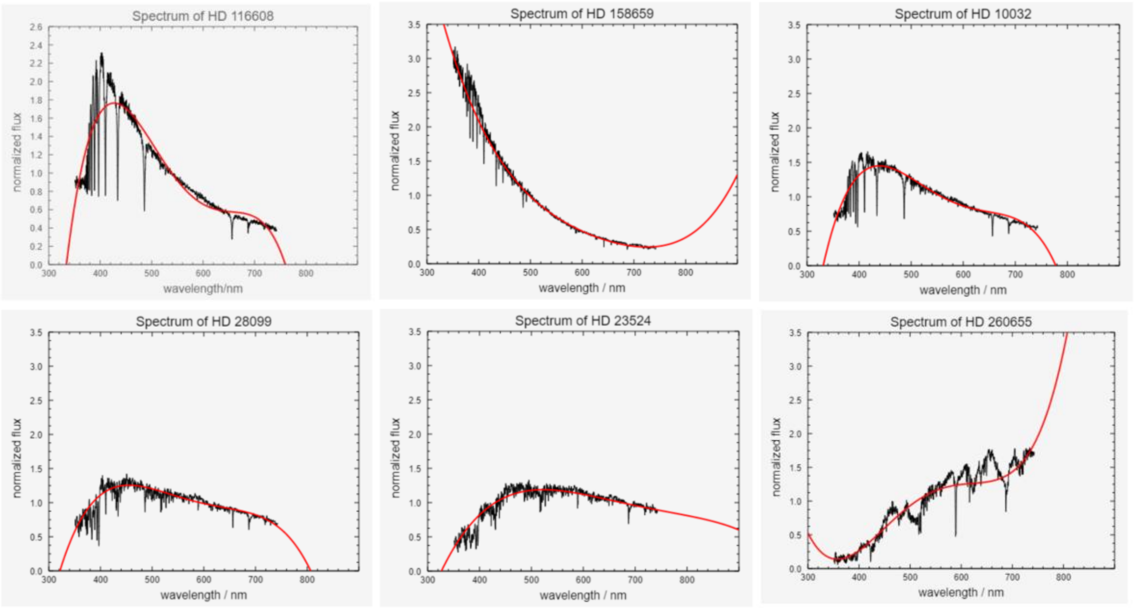


```
set PlotPad line properties style: continuous
width: 2 color: 255 0 0 on PlotPad
add graph interpolation polynomial for normalized data to PlotPad
PlotPad
```

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



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



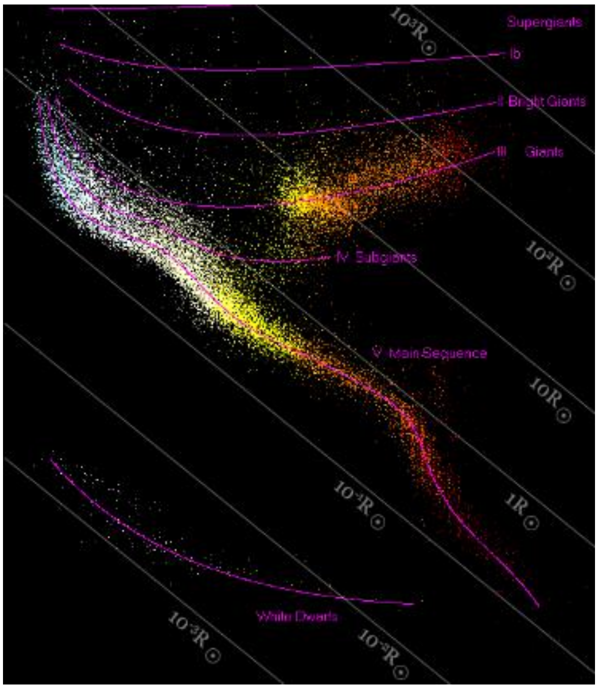
If you feed a neural network with the polynomial coefficients, it quickly learns to roughly assign a diagram to a star type. The program can "learn" on the basis of the old data which parameter intervals belong to which star classes. If one enters the data of a new star, then it determines the coefficients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

<u>Tasks</u>:

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.

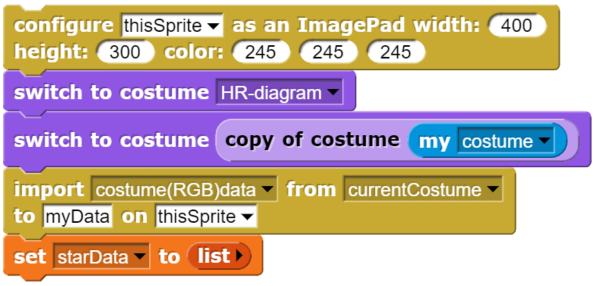
5.17 Classification with the kNN method

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])



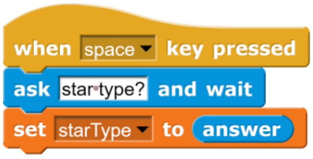
We want to classify new stars in this diagram using the *k-nearest neighbor (kNN)* method: We generate as training data a list of stars with their coordinates (simply as image coordinates in the diagram) and their type. If we want to classify a new star, we determine its position in the diagram and find 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.

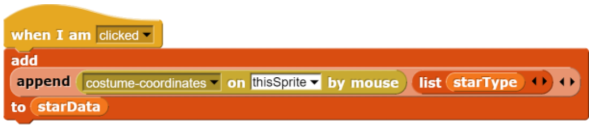


```
configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
switch to costume HR-diagram
switch to costume copy of costume my costume
import costume(RGB)data from currentCostume
to myData on thisSprite
set starData to list
```

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

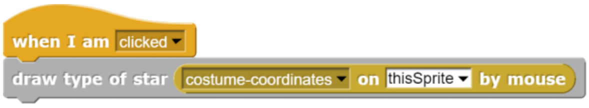


```
when space key pressed
ask star type? and wait
set starType to answer
```



```
when I am clicked
add
append costume-coordinates on thisSprite by mouse list starType
to starData
```

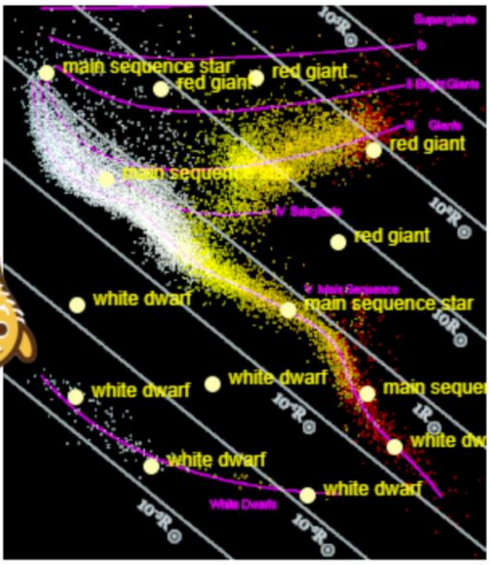
After that we can classify new stars by clicking and labeling them.

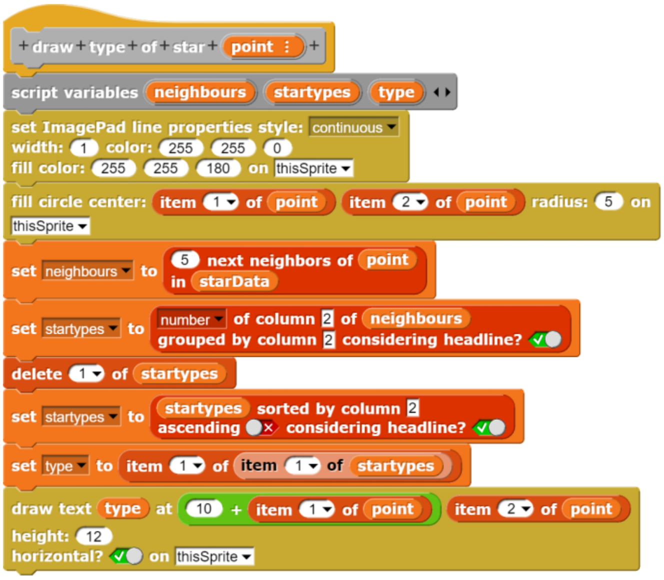


```
when I am clicked
draw type of star costume-coordinates on thisSprite by mouse
```

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

The result:

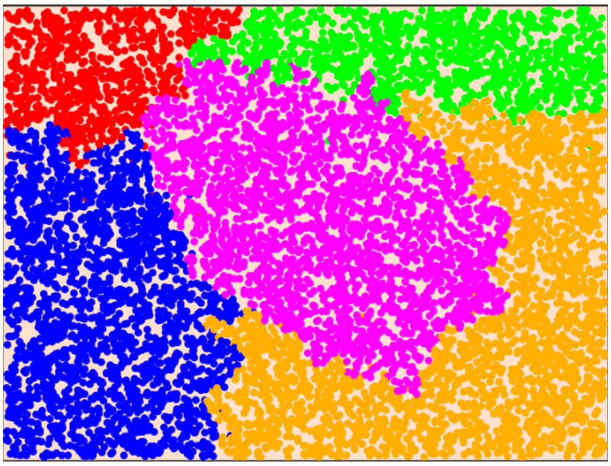




```
+draw+type+of+star+ (point :) +
script variables (neighbours) (startypes) (type) ◂▸
set ImagePad line properties style: [continuous ▾]
width: (1) color: (255) (255) (0)
fill color: (255) (255) (180) on [thisSprite ▾]
fill circle center: (item (1 ▾) of (point)) (item (2 ▾) of (point)) radius: (5)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

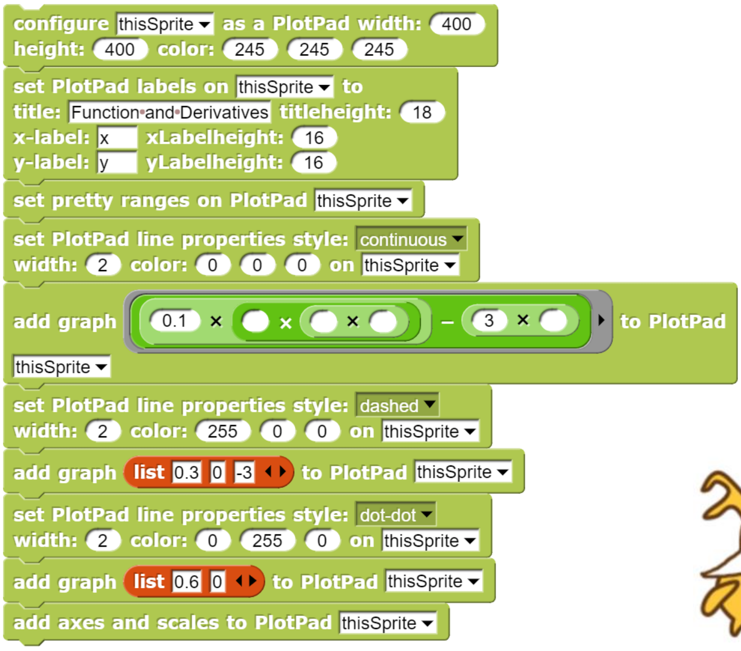
Tasks:

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

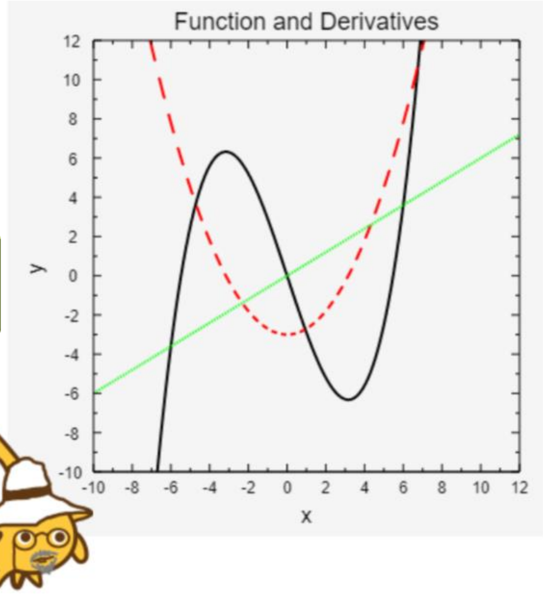


5.21 Drawing a function and its derivatives

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



```
configure thisSprite as a PlotPad width: 400
height: 400 color: 245 245 245
set PlotPad labels on thisSprite to
title: Function and Derivatives 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 × ( × ( × ))) − (3 × )) to PlotPad
thisSprite
set PlotPad line properties style: dashed
width: 2 color: 255 0 0 on thisSprite
add graph list 0.3 0 -3 to PlotPad thisSprite
set PlotPad line properties style: dot-dot
width: 2 color: 0 255 0 on thisSprite
add graph list 0.6 0 to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```

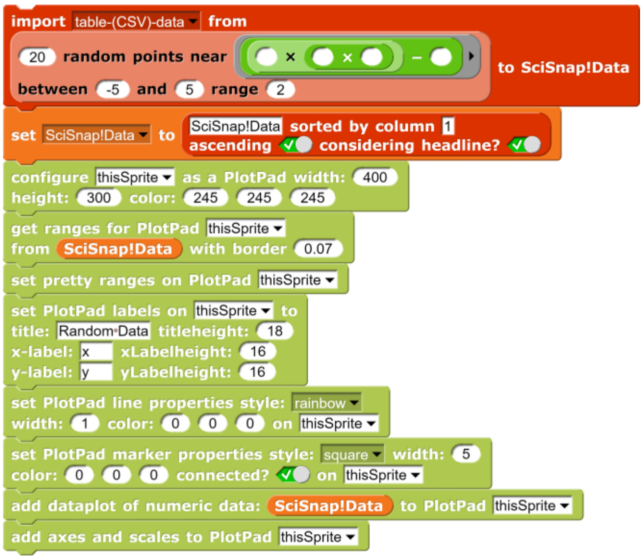


Tasks:

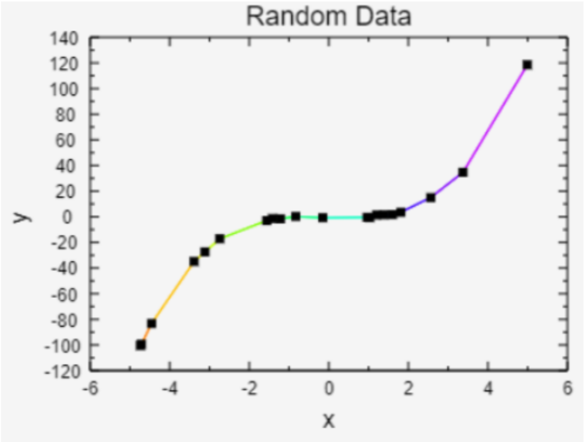
1. Plot different types of functions (trigonometric functions, logarithms, polynomials, ...) as graphs on a *PlotPad*.
2. Complete the function graphs with their derivatives.
3. Select different ranges of values, precision of the number representation and text sizes and labels for the representations.

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.

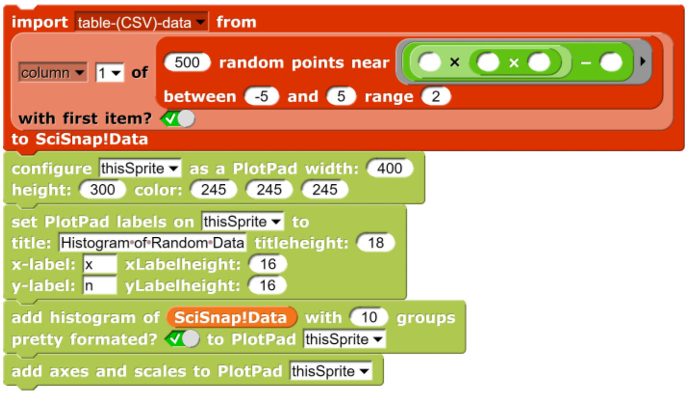


```
import table-(CSV)-data from
  20 random points near ( ( ) × ( ) × ( ) ) − ( )
  between -5 and 5 range 2                        to SciSnap!Data
set SciSnap!Data to SciSnap!Data sorted by column 1
                    ascending ✓ considering headline? ✓
configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245
get ranges for PlotPad thisSprite
from SciSnap!Data with border 0.07
set pretty ranges on PlotPad thisSprite
set PlotPad labels on thisSprite to
title: Random Data titleheight: 18
x-label: x xLabelheight: 16
y-label: y yLabelheight: 16
set PlotPad line properties style: rainbow
width: 1 color: 0 0 0 on thisSprite
set PlotPad marker properties style: square width: 5
color: 0 0 0 connected? ✓ on thisSprite
add dataplot of numeric data: SciSnap!Data to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```

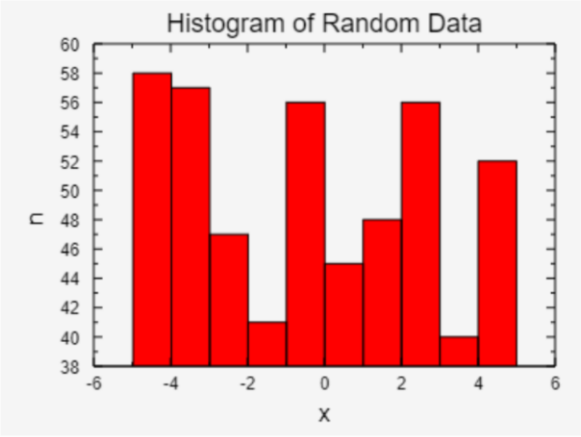


5.23 Histogram of random values

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.

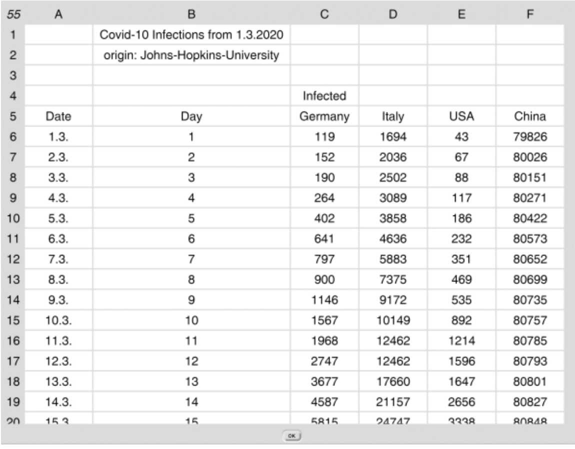


```
import table-(CSV)-data from
    column 1 of 500 random points near (( ) × ( ) × ( )) − ( )
                between -5 and 5 range 2
    with first item? ✓
to SciSnap!Data
configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on thisSprite to
title: Histogram of Random Data titleheight: 18
x-label: x xLabelheight: 16
y-label: n yLabelheight: 16
add histogram of SciSnap!Data with 10 groups
pretty formated? ✓ to PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```



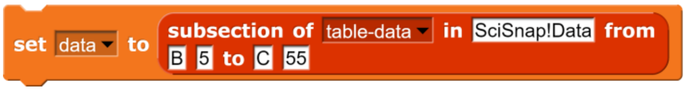
5.24 Covid-19 data analysis

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:

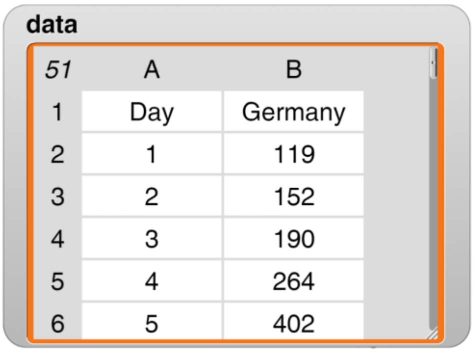


55	A	B	C	D	E	F
1		Covid-10 Infections from 1.3.2020				
2		origin: Johns-Hopkins-University				
3						
4			Infected			
5	Date	Day	Germany	Italy	USA	China
6	1.3.	1	119	1694	43	79826
7	2.3.	2	152	2036	67	80026
8	3.3.	3	190	2502	88	80151
9	4.3.	4	264	3089	117	80271
10	5.3.	5	402	3858	186	80422
11	6.3.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

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



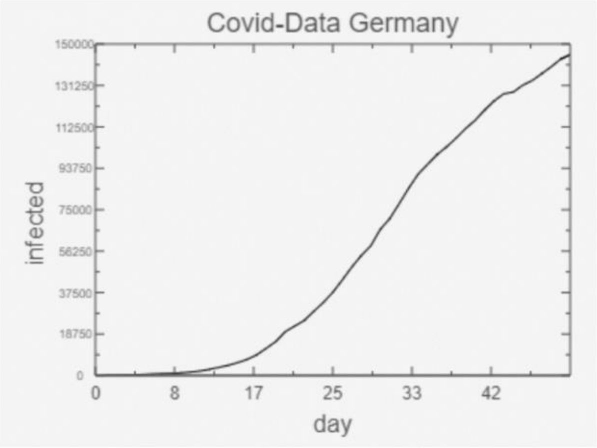
```
set data to subsection of table-data in SciSnap!Data from
B 5 to C 55
```

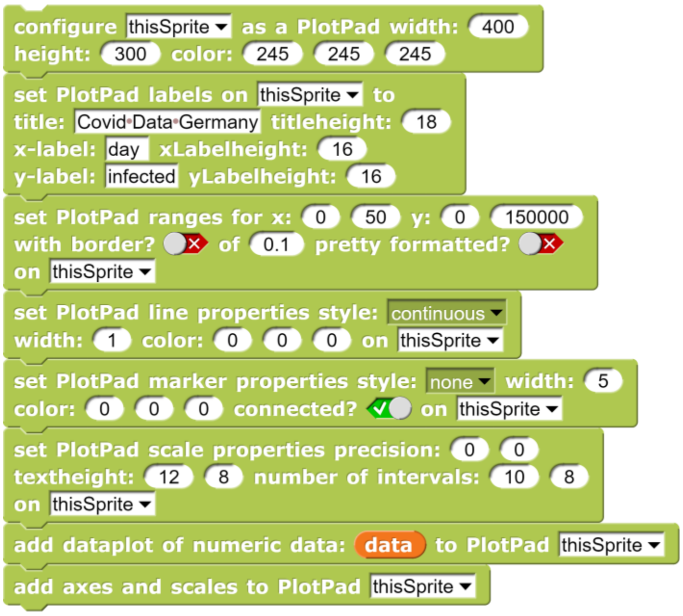


data

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

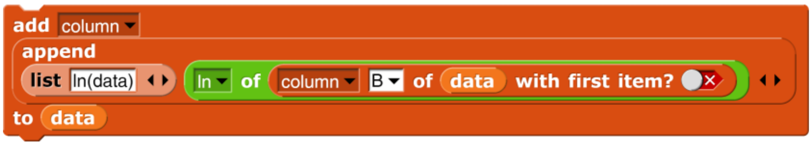
Let's first get an overview of this:





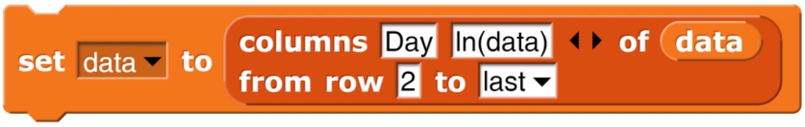
```
configure thisSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on thisSprite to
title: Covid Data Germany titleheight: 18
x-label: day xLabelheight: 16
y-label: infected yLabelheight: 16
set PlotPad ranges for x: 0 50 y: 0 150000
with border? ☒ of 0.1 pretty formatted?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

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



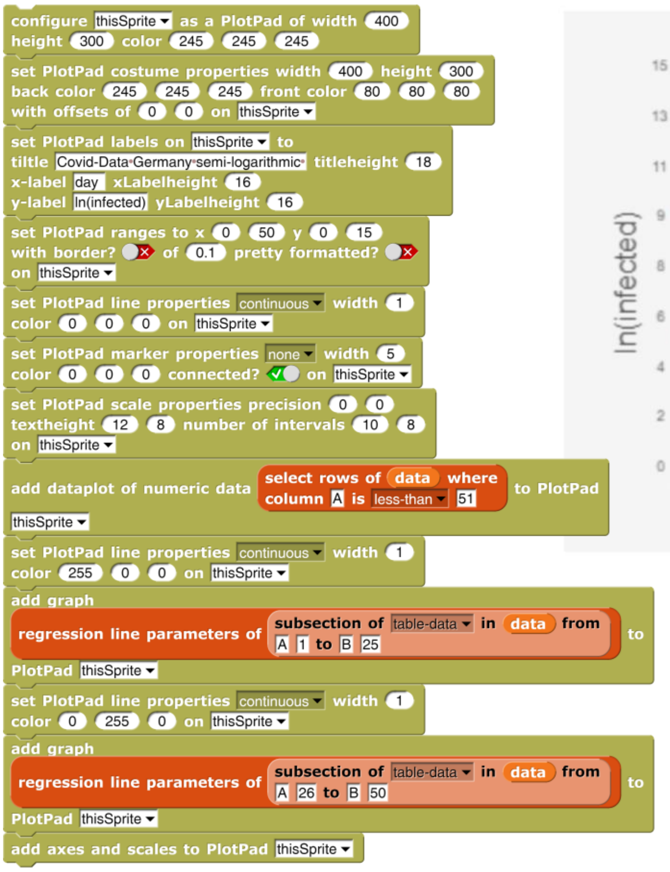
```
add column
append
list ln(data) ◂▸
ln of column B of data with first item? ☒ ◂▸
to data
```

… pick out the two interesting columns …

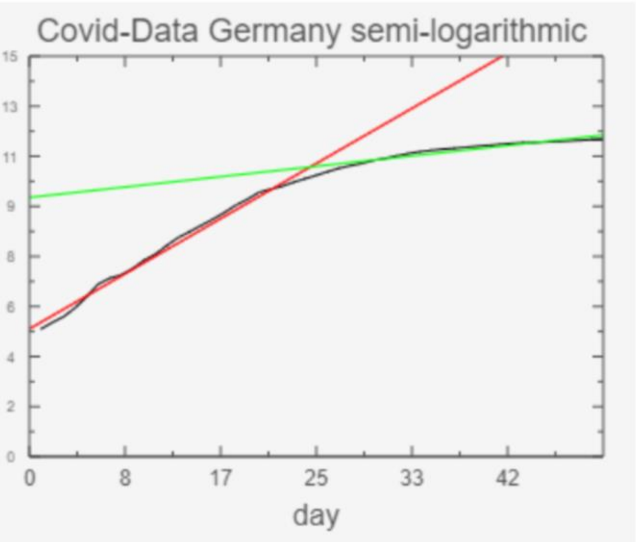


```
set data to columns Day ln(data) ◂▸ of data
from row 2 to last
```

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



```
configure thisSprite as a PlotPad of width 400
height 300 color 245 245 245
set PlotPad costume properties width 400 height 300
back color 245 245 245 front color 80 80 80
with offsets of 0 0 on thisSprite
set PlotPad labels on thisSprite to
tiltle Covid-Data·Germany·semi-logarithmic· titleheight 18
x-label day xLabelheight 16
y-label ln(infected) yLabelheight 16
set PlotPad ranges to x 0 50 y 0 15
with border? ☒ of 0.1 pretty formatted? ☒
on thisSprite
set PlotPad line properties continuous width 1
color 0 0 0 on thisSprite
set PlotPad marker properties none width 5
color 0 0 0 connected? ✓ on thisSprite
set PlotPad scale properties precision 0 0
textheight 12 8 number of intervals 10 8
on thisSprite
add dataplot of numeric data (select rows of (data) where column A is less-than 51) to PlotPad
thisSprite
set PlotPad line properties continuous width 1
color 255 0 0 on thisSprite
add graph
  (regression line parameters of (subsection of table-data in (data) from A 1 to B 25)) to
PlotPad thisSprite
set PlotPad line properties continuous width 1
color 0 255 0 on thisSprite
add graph
  (regression line parameters of (subsection of table-data in (data) from A 26 to B 50)) to
PlotPad thisSprite
add axes and scales to PlotPad thisSprite
```



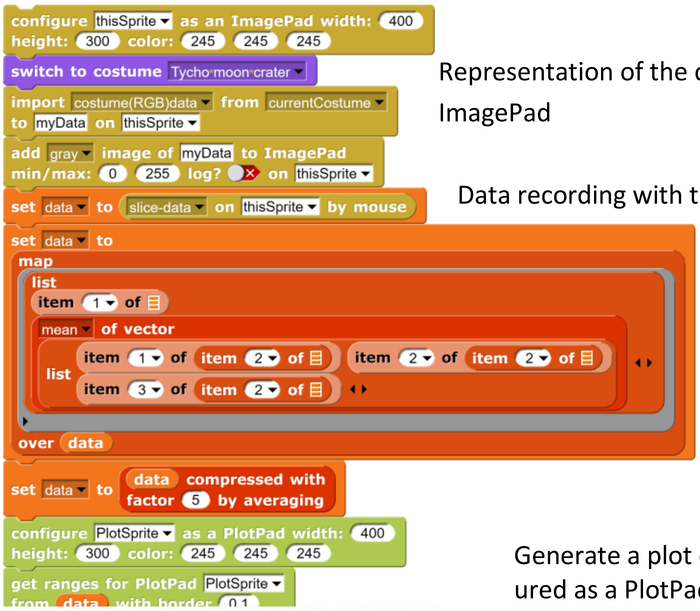
That's when hope arose!

Tasks:

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

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.



```
configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
switch to costume Tycho moon crater
import costume(RGB)data from currentCostume
to myData on thisSprite
add gray image of myData to ImagePad
min/max: 0 255 log? on thisSprite
```

Representation of the crater image on the ImagePad

```
set data to slice-data on thisSprite by mouse
```

Data recording with the mouse

```
set data to
map
  list
    item 1 of ▤
    mean of vector
      list
        item 1 of item 2 of ▤   item 2 of item 2 of ▤
        item 3 of item 2 of ▤
over data
```

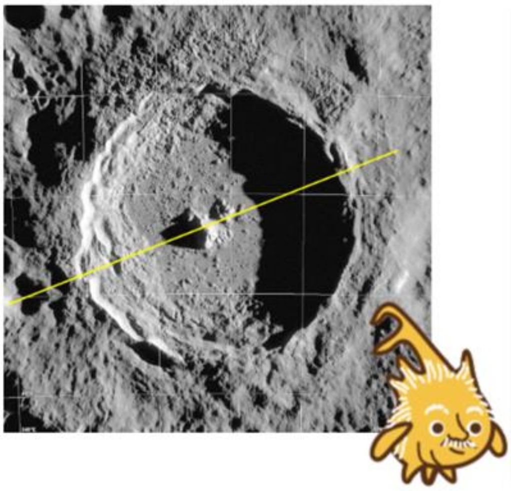
Conversion of RGB values to gray values

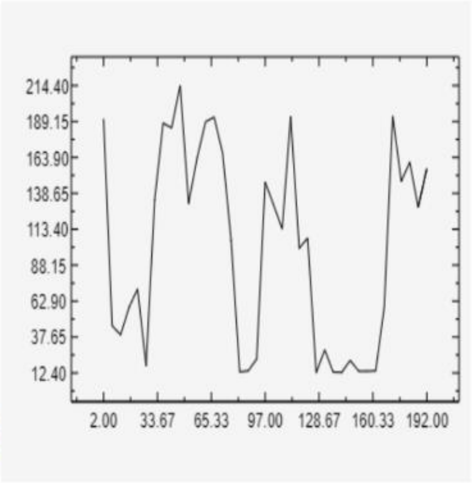
```
set data to data compressed with factor 5 by averaging
```

Data compression with a factor of 5

```
configure PlotSprite as a PlotPad width: 400
height: 300 color: 245 245 245
get ranges for PlotPad PlotSprite
from data with border 0.1
```

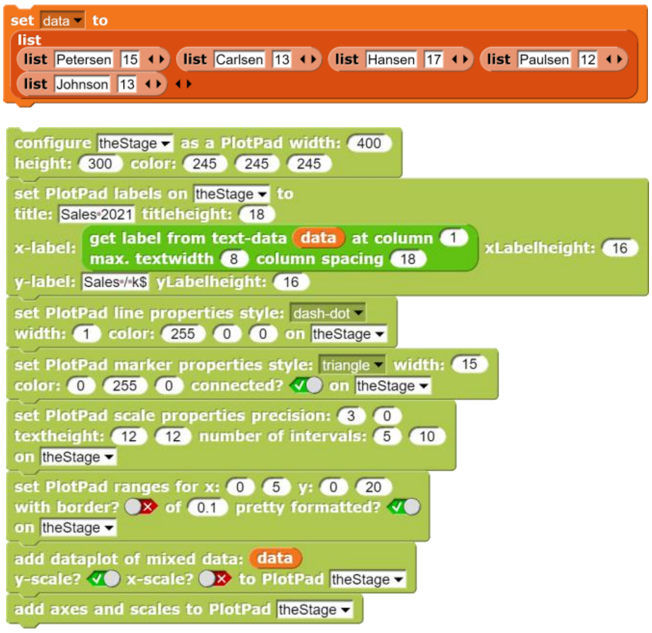
Generate a plot on a second sprite configured as a PlotPad.





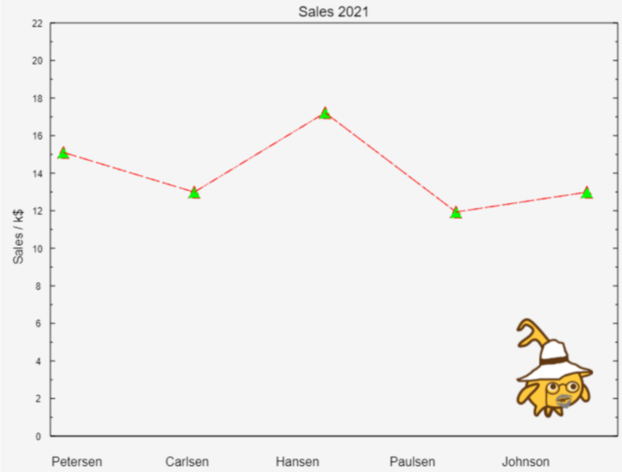
5.26 Plot of mixed data

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*.



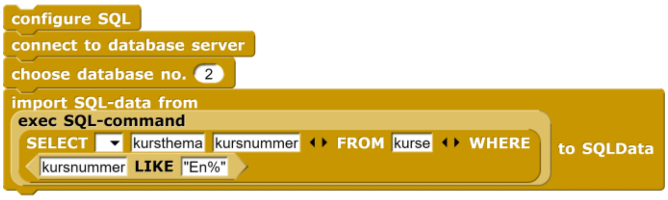
```
set data ▾ to
  list
    list Petersen 15 ◂▸   list Carlsen 13 ◂▸   list Hansen 17 ◂▸   list Paulsen 12 ◂▸
    list Johnson 13 ◂▸ ◂▸

configure theStage ▾ as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on theStage ▾ to
title: Sales 2021 titleheight: 18
x-label: get label from text-data data at column 1        xLabelheight: 16
         max. textwidth 8 column spacing 18
y-label: Sales / k$ yLabelheight: 16
set PlotPad line properties style: dash-dot ▾
width: 1 color: 255 0 0 on theStage ▾
set PlotPad marker properties style: triangle ▾ width: 15
color: 0 255 0 connected? ✓ on theStage ▾
set PlotPad scale properties precision: 3 0
textheight: 12 12 number of intervals: 5 10
on theStage ▾
set PlotPad ranges for x: 0 5 y: 0 20
with border? ✗ of 0.1 pretty formatted? ✓
on theStage ▾
add dataplot of mixed data: data
y-scale? ✓ x-scale? ✗ to PlotPad theStage ▾
add axes and scales to PlotPad theStage ▾
```

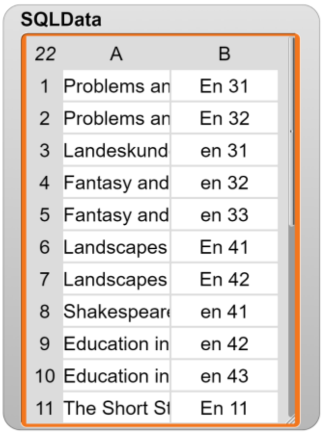


5.27 A simple SQL query

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 ▾ kursthema kursnummer ◂▸ FROM kurse ◂▸ WHERE
  kursnummer LIKE "En%"
                                                  to SQLData
```

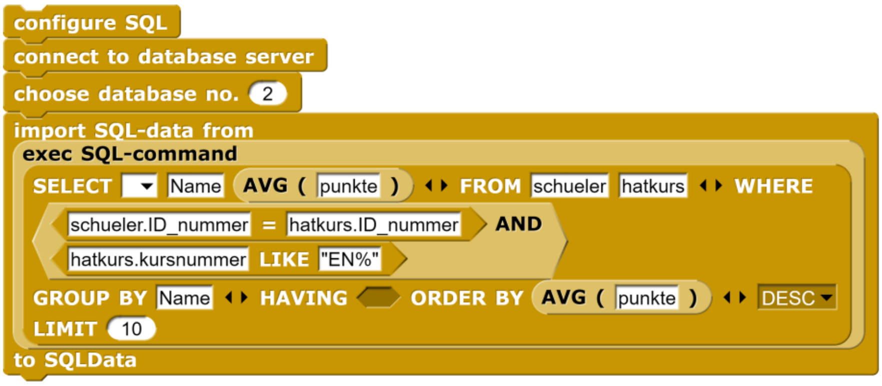


SQLData

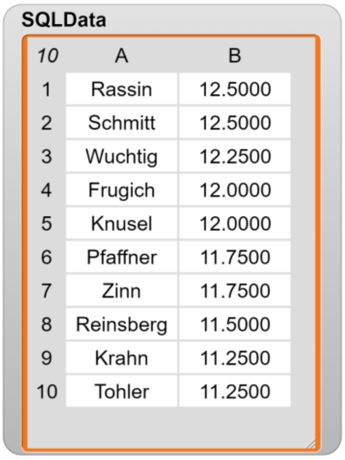
22	A	B
1	Problems an	En 31
2	Problems an	En 32
3	Landeskund	en 31
4	Fantasy and	en 32
5	Fantasy and	en 33
6	Landscapes	En 41
7	Landscapes	En 42
8	Shakespeare	en 41
9	Education in	en 42
10	Education in	en 43
11	The Short St	En 11

5.28 A more complex SQL query

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



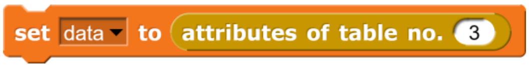
```
configure SQL
connect to database server
choose database no. 2
import SQL-data from
 exec SQL-command
  SELECT ▾ Name AVG ( punkte ) ◂▸ FROM schueler hatkurs ◂▸ WHERE
   schueler.ID_nummer = hatkurs.ID_nummer AND
   hatkurs.kursnummer LIKE "EN%"
  GROUP BY Name ◂▸ HAVING ORDER BY AVG ( punkte ) ◂▸ DESC ▾
  LIMIT 10
to SQLData
```



SQLData

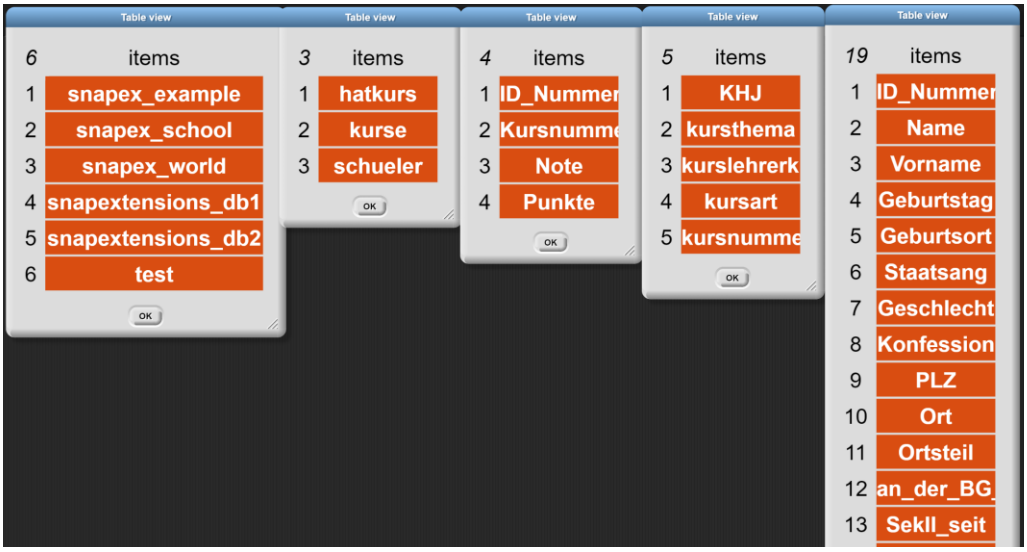
10	A	B
1	Rassin	12.5000
2	Schmitt	12.5000
3	Wuchtig	12.2500
4	Frugich	12.0000
5	Knusel	12.0000
6	Pfaffner	11.7500
7	Zinn	11.7500
8	Reinsberg	11.5000
9	Krahn	11.2500
10	Tohler	11.2500

5.30 Dealing with the SQL library



```
set data to attributes of table no. 3
```

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.



6	items
1	snapex_example
2	snapex_school
3	snapex_world
4	snapextensions_db1
5	snapextensions_db2
6	test

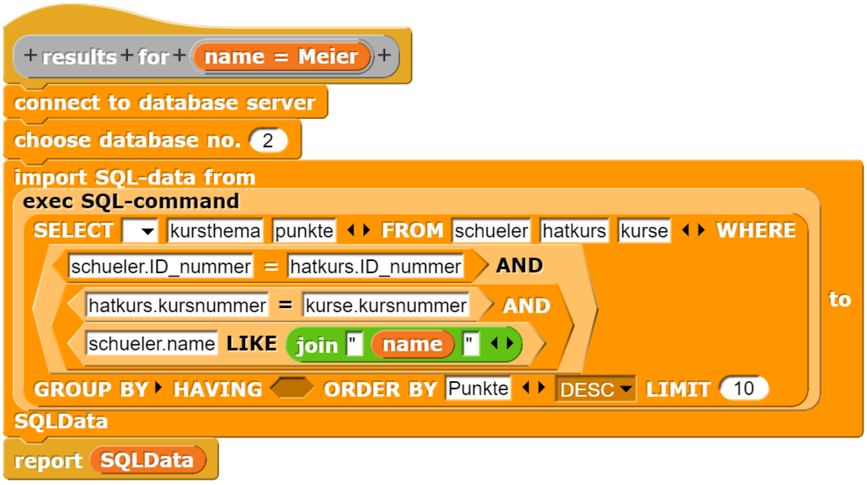
3	items
1	hatkurs
2	kurse
3	schueler

4	items
1	ID_Nummer
2	Kursnumme
3	Note
4	Punkte

5	items
1	KHJ
2	kursthema
3	kurslehrerk
4	kursart
5	kursnumme

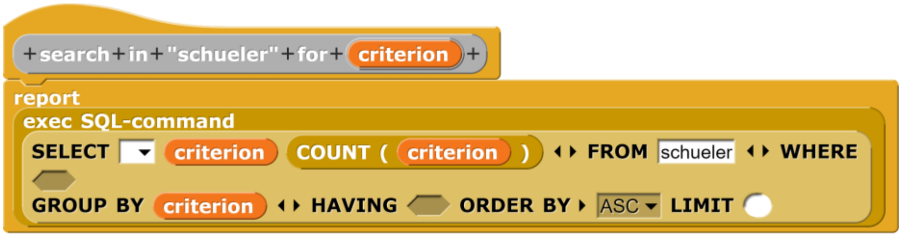
19	items
1	ID_Nummer
2	Name
3	Vorname
4	Geburtstag
5	Geburtsort
6	Staatsang
7	Geschlecht
8	Konfession
9	PLZ
10	Ort
11	Ortsteil
12	an_der_BG
13	SekII_seit

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



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

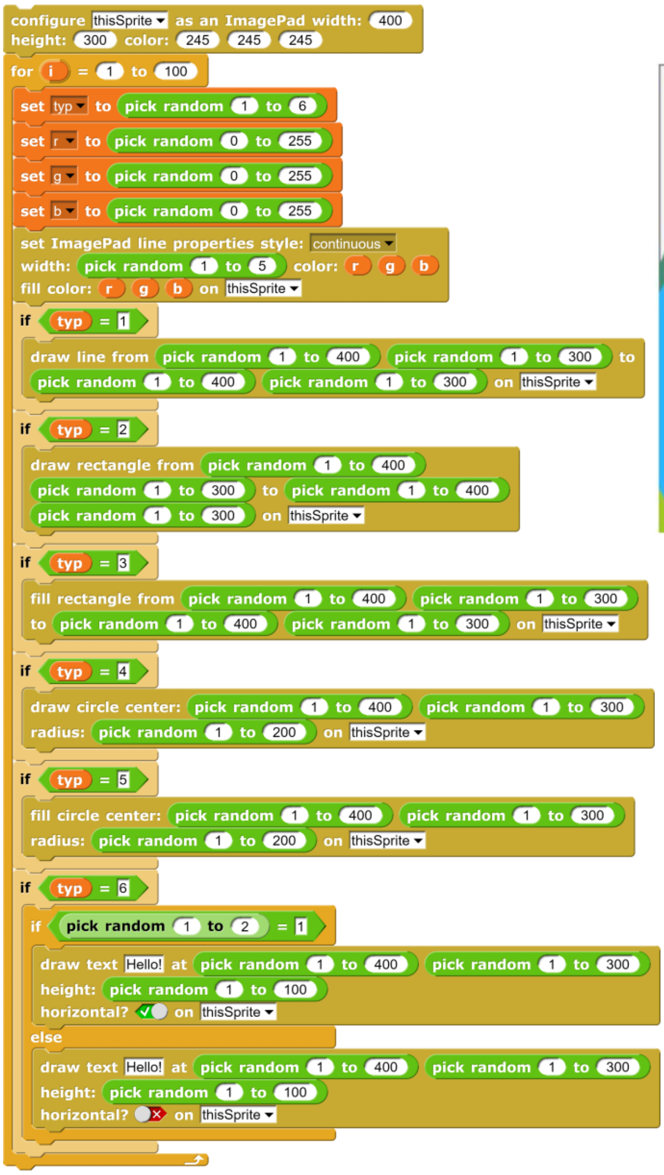
Example: For statistical purposes, the *schueler* table should be searchable by different criteria.



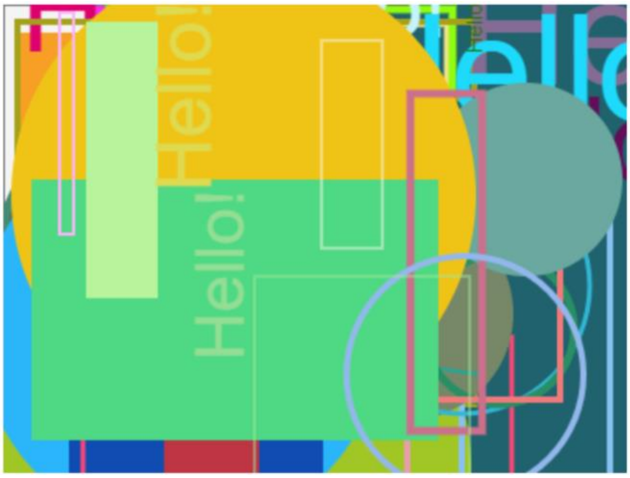
```
+ search + in + "schueler" + for + criterion +
report
  exec SQL-command
    SELECT [ ▾] criterion COUNT ( criterion ) ◂▸ FROM schueler ◂▸ WHERE
    GROUP BY criterion ◂▸ HAVING ORDER BY ▸ ASC ▾ LIMIT
```

5.31 Simple random graphic

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

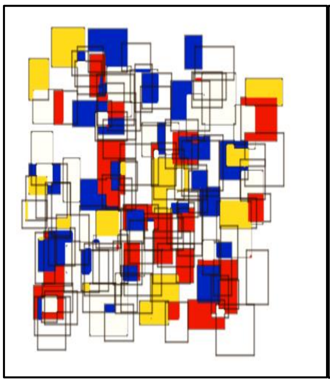


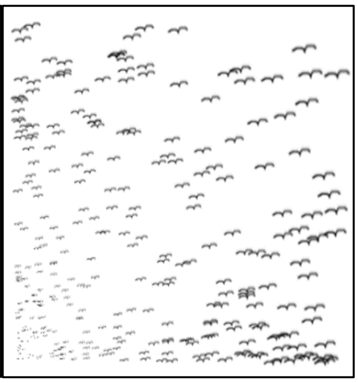
```
configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
for i = 1 to 100
    set typ to pick random 1 to 6
    set r to pick random 0 to 255
    set g to pick random 0 to 255
    set b to pick random 0 to 255
    set ImagePad line properties style: continuous
    width: pick random 1 to 5 color: r g b
    fill color: r g b on thisSprite
    if typ = 1
        draw line from pick random 1 to 400 pick random 1 to 300 to
        pick random 1 to 400 pick random 1 to 300 on thisSprite
    if typ = 2
        draw rectangle from pick random 1 to 400
        pick random 1 to 300 to pick random 1 to 400
        pick random 1 to 300 on thisSprite
    if typ = 3
        fill rectangle from pick random 1 to 400 pick random 1 to 300
        to pick random 1 to 400 pick random 1 to 300 on thisSprite
    if typ = 4
        draw circle center: pick random 1 to 400 pick random 1 to 300
        radius: pick random 1 to 200 on thisSprite
    if typ = 5
        fill circle center: pick random 1 to 400 pick random 1 to 300
        radius: pick random 1 to 200 on thisSprite
    if typ = 6
        if pick random 1 to 2 = 1
            draw text Hello! at pick random 1 to 400 pick random 1 to 300
            height: pick random 1 to 100
            horizontal? ✓ on thisSprite
        else
            draw text Hello! at pick random 1 to 400 pick random 1 to 300
            height: pick random 1 to 100
            horizontal? ✗ on thisSprite
```



Tasks:

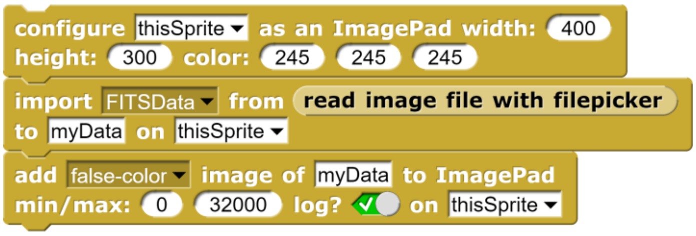
1. Search the web for images by *Piet Mondrian*. Try to create similar random images on the *ImagePad*.
2. Using a "vanishing point", you can create images in which objects appear to move "from back to front". Try it.



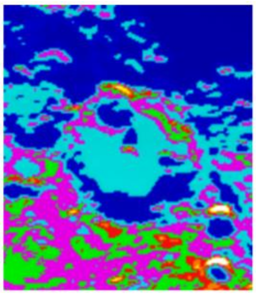


5.32 False color image of a lunar crater

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

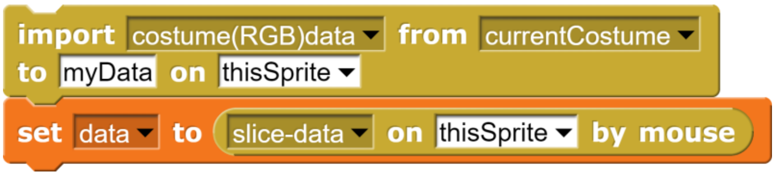


```
configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
import FITSData from read image file with filepicker
to myData on thisSprite
add false-color image of myData to ImagePad
min/max: 0 32000 log? ✓ on thisSprite
```

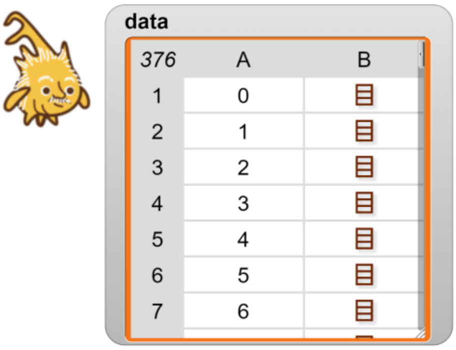


5.33 Section through an image of the lunar crater Tycho

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

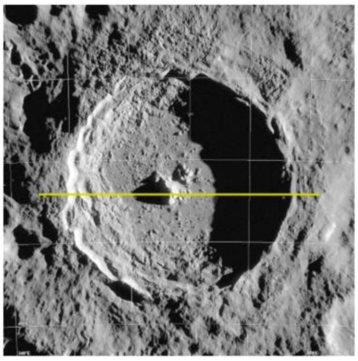


```
import costume(RGB)data from currentCostume
to myData on thisSprite
set data to slice-data on thisSprite by mouse
```



data

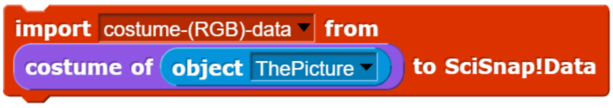
376	A	B
1	0	▤
2	1	▤
3	2	▤
4	3	▤
5	4	▤
6	5	▤
7	6	▤



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*".

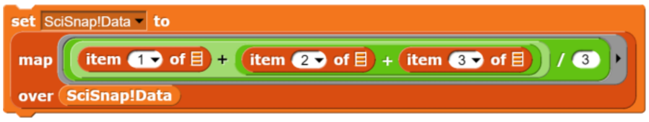
First of all we load the image into the data area *SciSnap!Data*:



```
import costume-(RGB)-data from
costume of object ThePicture to SciSnap!Data
```

We obtain 120,000 RBG values.

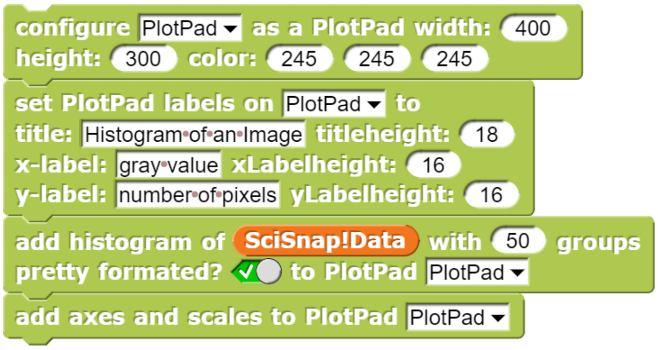
We convert these into gray values.



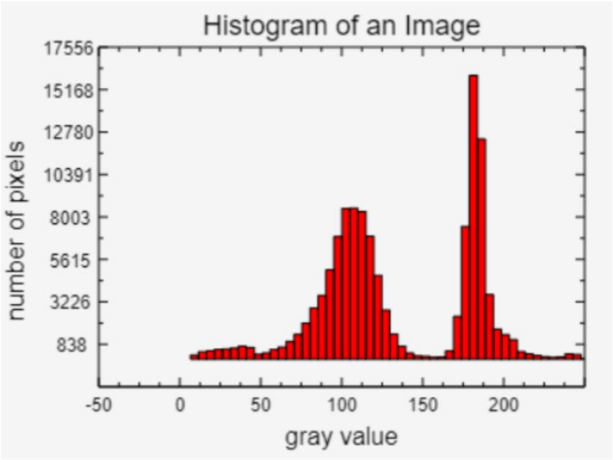
```
set SciSnap!Data to
map ((item 1 of ⌸ + item 2 of ⌸ + item 3 of ⌸) / 3)
over SciSnap!Data
```

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*.



```
configure PlotPad as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on PlotPad to
title: Histogram of an Image titleheight: 18
x-label: gray value xLabelheight: 16
y-label: number of pixels yLabelheight: 16
add histogram of SciSnap!Data with 50 groups
pretty formated? ✓ to PlotPad PlotPad
add axes and scales to PlotPad PlotPad
```

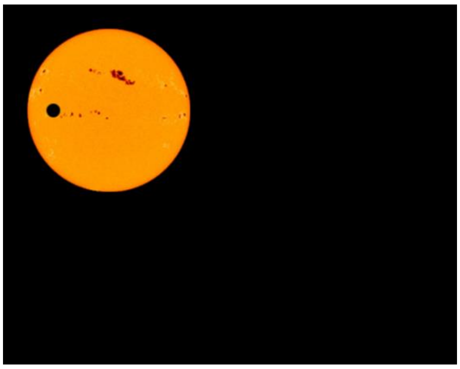


Tasks:

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.

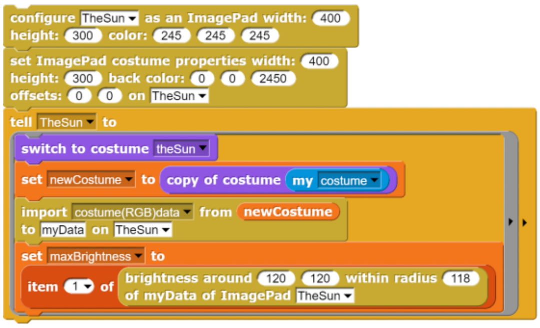
5.37 Simulation of a planetary transit in front of the sun

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.



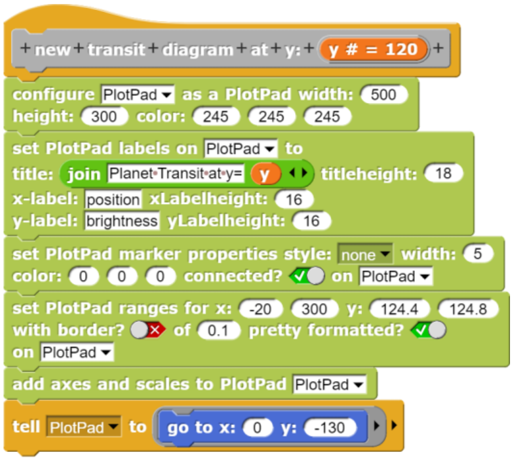
The planet should pass in front of the sun as a black circle. When we draw such a circle, we change the actual image of the sun. Therefore, we draw a copy *newCostume* from this image, on which we draw after that. Our planet should move from the very left a little bit outside of the image (*x=-2r*) to 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*.

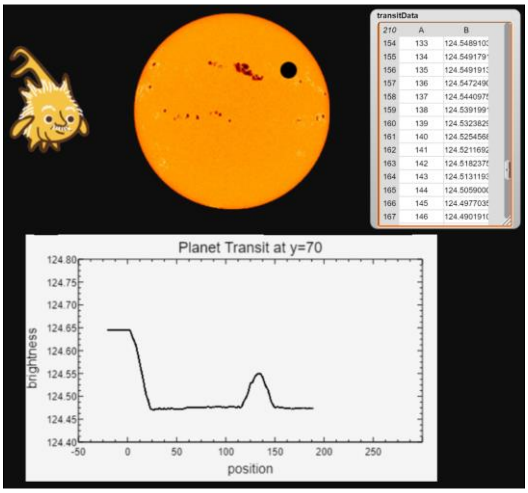


```
configure TheSun as an ImagePad width: 400
height: 300 color: 245 245 245
set ImagePad costume properties width: 400
height: 300 back color: 0 0 2450
offsets: 0 0 on TheSun
tell TheSun to
    switch to costume theSun
    set newCostume to copy of costume my costume
    import costume(RGB)data from newCostume
    to myData on TheSun
    set maxBrightness to
        item 1 of brightness around 120 120 within radius 118
                  of myData of ImagePad TheSun
```

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*.



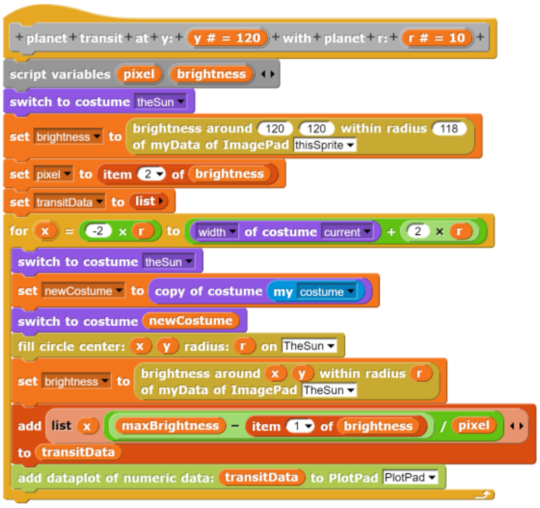
```
+new+transit+diagram+at+y:+ y # = 120 +
configure PlotPad as a PlotPad width: 500
height: 300 color: 245 245 245
set PlotPad labels on PlotPad to
title: join Planet·Transit·at·y= y titleheight: 18
x-label: position xLabelheight: 16
y-label: brightness yLabelheight: 16
set PlotPad marker properties style: none width: 5
color: 0 0 0 connected? ✓ on PlotPad
set PlotPad ranges for x: -20 300 y: 124.4 124.8
with border? ✗ of 0.1 pretty formatted? ✓
on PlotPad
add axes and scales to PlotPad PlotPad
tell PlotPad to go to x: 0 y: -130
```



transitData

210	A	B
154	133	124.5489103
155	134	124.5491791
156	135	124.5491913
157	136	124.5472490
158	137	124.5440975
159	138	124.5391991
160	139	124.5323829
161	140	124.5254568
162	141	124.5211692
163	142	124.5182375
164	143	124.5131193
165	144	124.5059000
166	145	124.4977035
167	146	124.4901910

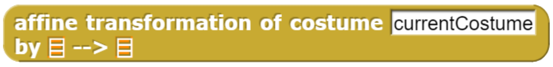
Then we write a block that determines the brightness data as described and refreshes the diagram in parallel. The result corresponds roughly to one of the methods used to find exoplanets.



```
+planet+transit+at+y:+ y # = 120 +with+planet+r:+ r # = 10 +
script variables pixel brightness
switch to costume theSun
set brightness to brightness around 120 120 within radius 118
                  of myData of ImagePad thisSprite
set pixel to item 2 of brightness
set transitData to list
for x = -2 × r to width of costume current + 2 × r
    switch to costume theSun
    set newCostume to copy of costume my costume
    switch to costume newCostume
    fill circle center: x y radius: r on TheSun
    set brightness to brightness around x y within radius r
                      of myData of ImagePad TheSun
    add list x ((maxBrightness − item 1 of brightness) / pixel)
    to transitData
    add dataplot of numeric data: transitData to PlotPad PlotPad
```

5.38 Affine transformation of an image

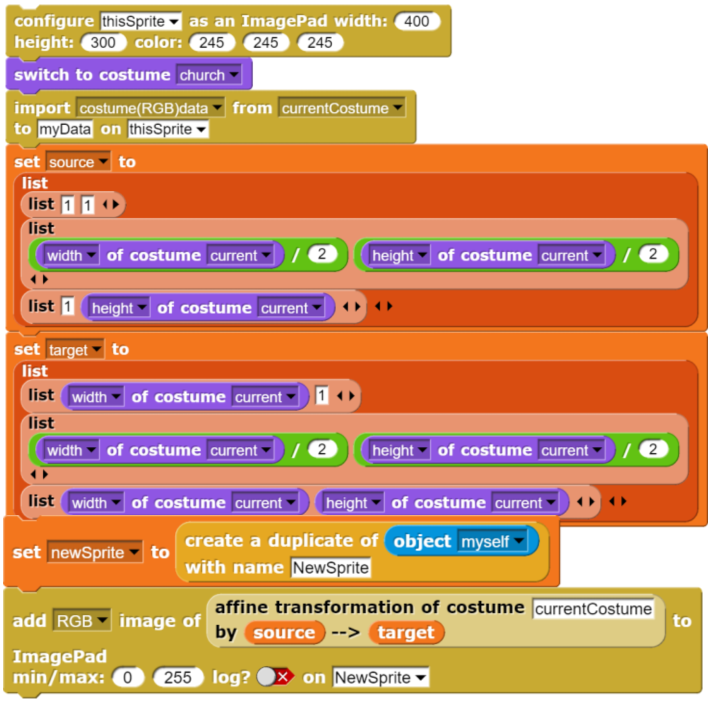
In the *SciSnap!ImagePadLibrary* we find a block that allows to make affine transformations in an image by mapping three points to three others - and all other points accordingly. The current costume of a sprite is taken as the image.



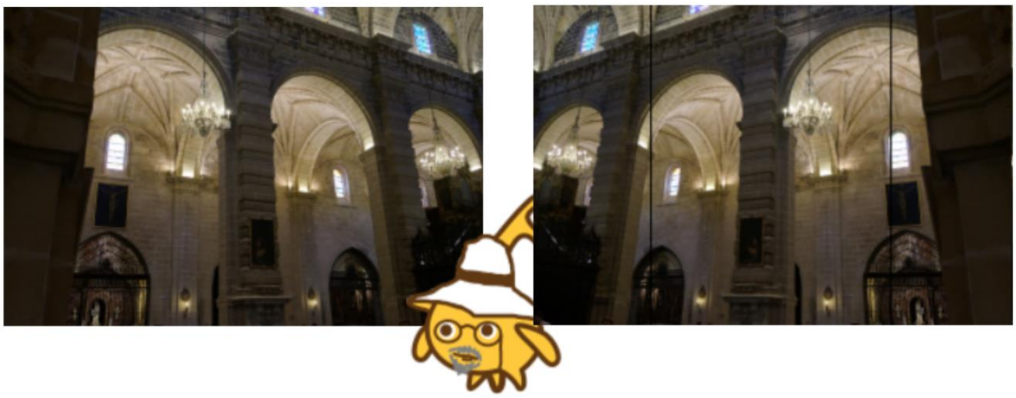
```
affine transformation of costume currentCostume
by ⌸ --> ⌸
```

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.



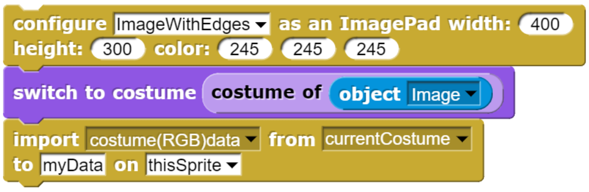
```
configure thisSprite as an ImagePad width: 400
height: 300 color: 245 245 245
switch to costume church
import costume(RGB)data from currentCostume
to myData on thisSprite
set source to
  list
    list 1 1
    list
      (width of costume current) / 2   (height of costume current) / 2
    list 1 (height of costume current)
set target to
  list
    list (width of costume current) 1
    list
      (width of costume current) / 2   (height of costume current) / 2
    list (width of costume current) (height of costume current)
set newSprite to create a duplicate of object myself
                 with name NewSprite
add RGB image of (affine transformation of costume currentCostume
                  by source --> target) to
ImagePad
min/max: 0 255 log? on NewSprite
```



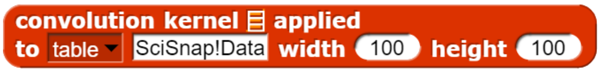
5.39 Kernel application for edge detection

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*.

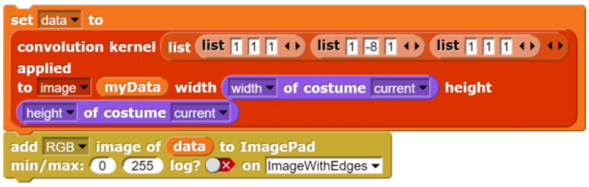
On this image data we apply the *Laplace* kernel $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ using the block *convolution kernel applied* of the *SciSnap!DataLibrary* and store the result in the variable *data*. We display the result as a new costume. For comparison, a second sprite represents the original image.



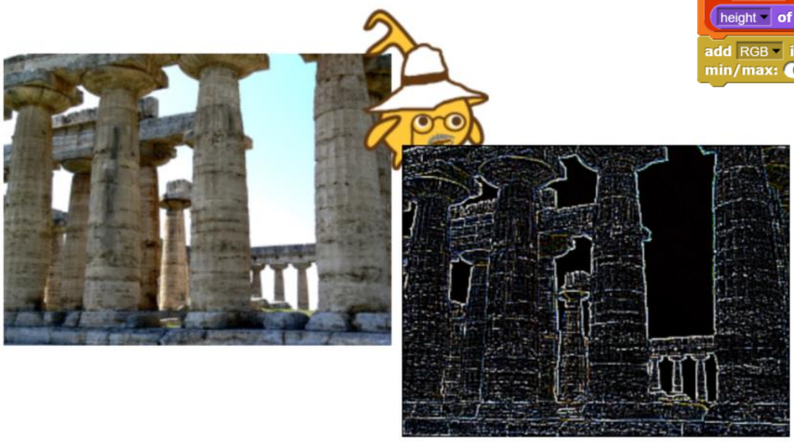
```
configure ImageWithEdges as an ImagePad width: 400
height: 300 color: 245 245 245
switch to costume (costume of (object Image))
import costume(RGB)data from currentCostume
to myData on thisSprite
```



```
convolution kernel ☰ applied
to table SciSnap!Data width 100 height 100
```



```
set data to
  convolution kernel (list (list 1 1 1) (list 1 -8 1) (list 1 1 1))
  applied
  to image (myData) width (width of costume current) height
  (height of costume current)
add RGB image of (data) to ImagePad
min/max: 0 255 log? ☒ on ImageWithEdges
```



Tasks:

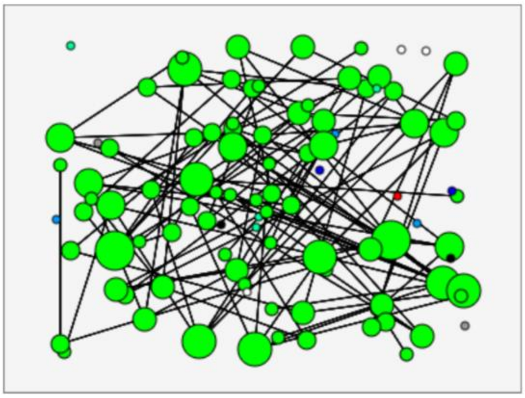
1. Images are sometimes a bit "flat". This is because they do not use the full range of values for the three color channels from 0 to 255.
   - a: Develop a method to determine and display the value range of an image.
   - b: Develop a method to exploit the full range of values, i.e. map black pixels 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.

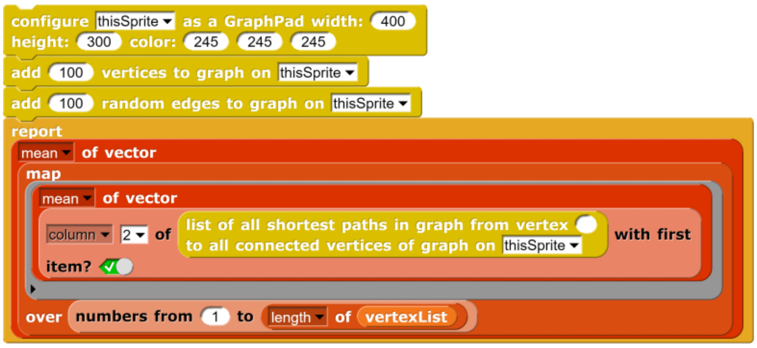
2. On pictures, you can try to find "faces" by highlighting related areas of a color range, e.g. "orange", and erasing the rest of the picture. Try to develop a new block for this.
   - b: Using a kernel, the edges of such areas can be isolated. Find out about suitable kernels on the web, for example, 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.

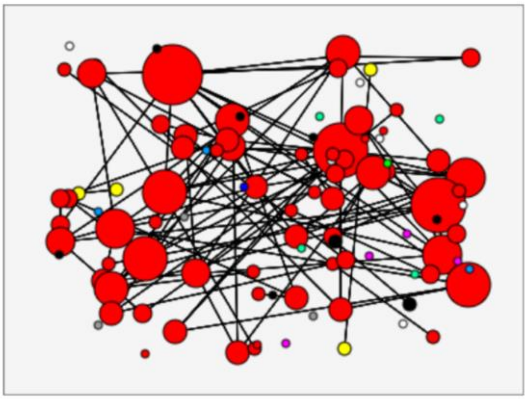


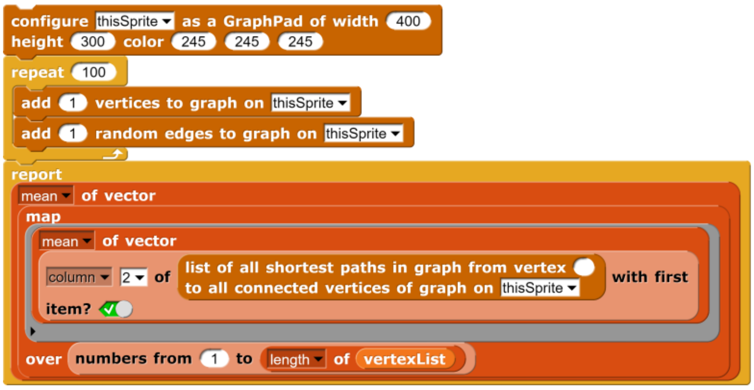


```
configure thisSprite as a GraphPad width: 400
height: 300 color: 245 245 245
add 100 vertices to graph on thisSprite
add 100 random edges to graph on thisSprite
report
  mean of vector
    map
      mean of vector
        column 2 of list of all shortest paths in graph from vertex ( )
                    to all connected vertices of graph on thisSprite  with first
        item? ✓
    over numbers from 1 to length of vertexList
```

5.41 Mean distance in a scalefree graph (Small Worlds)

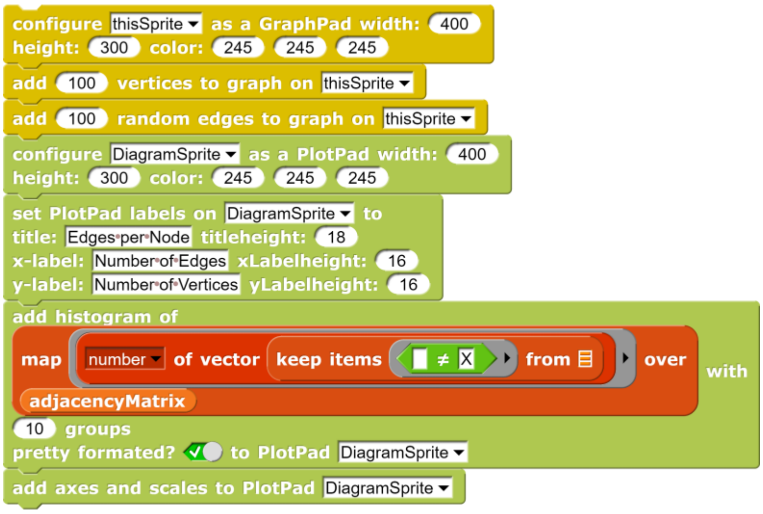
We compute the average distance between the vertices in a graph where alternating vertices and edges have been created.



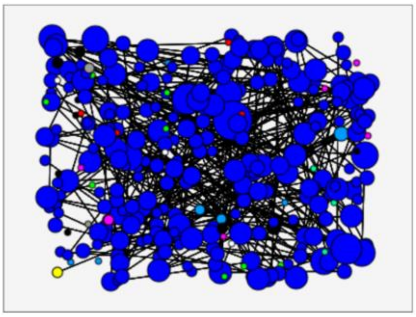


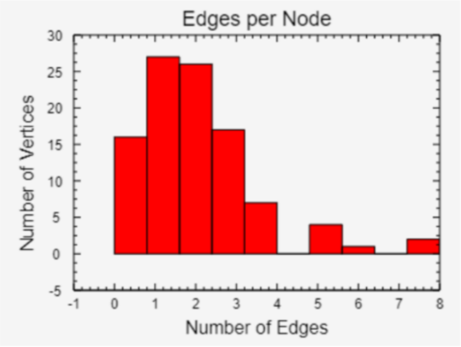
```
configure thisSprite as a GraphPad of width 400
height 300 color 245 245 245
repeat 100
  add 1 vertices to graph on thisSprite
  add 1 random edges to graph on thisSprite
report
  mean of vector
    map
      mean of vector
        column 2 of list of all shortest paths in graph from vertex ( )
                    to all connected vertices of graph on thisSprite  with first
        item? ✓
    over numbers from 1 to length of vertexList
```

5.42 "Edges per vertex" histogram in a random graph

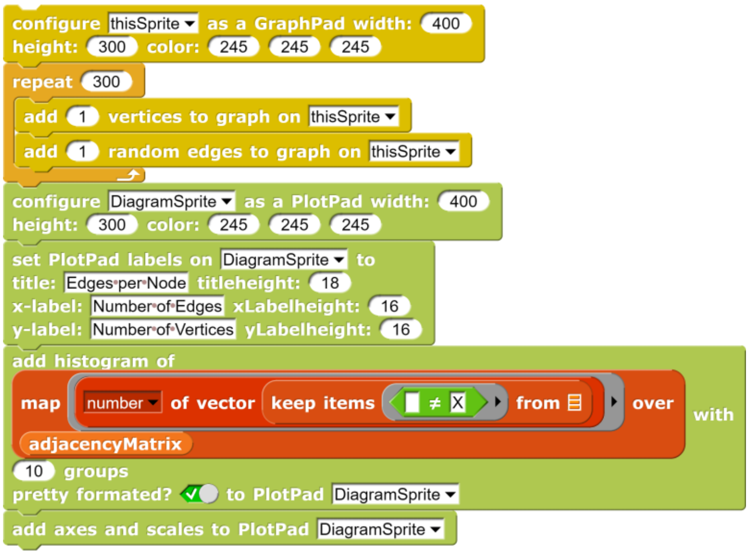


```
configure thisSprite as a GraphPad width: 400
height: 300 color: 245 245 245
add 100 vertices to graph on thisSprite
add 100 random edges to graph on thisSprite
configure DiagramSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on DiagramSprite to
title: Edges per Node titleheight: 18
x-label: Number of Edges xLabelheight: 16
y-label: Number of Vertices yLabelheight: 16
add histogram of
    map number of vector keep items ( ≠ X) from ⌸ over
    adjacencyMatrix
with
10 groups
pretty formated? ✓ to PlotPad DiagramSprite
add axes and scales to PlotPad DiagramSprite
```

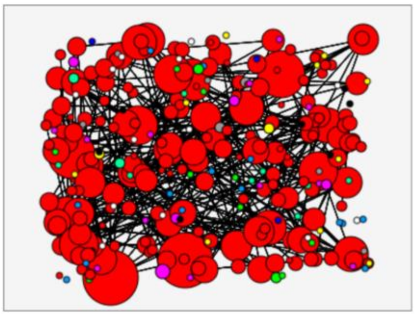


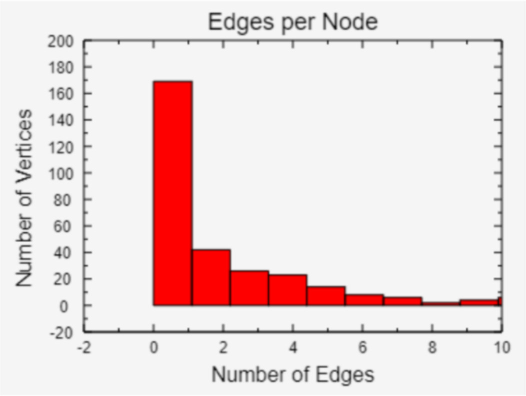


5.43 "Edges per vertex" histogram in a scalefree graph



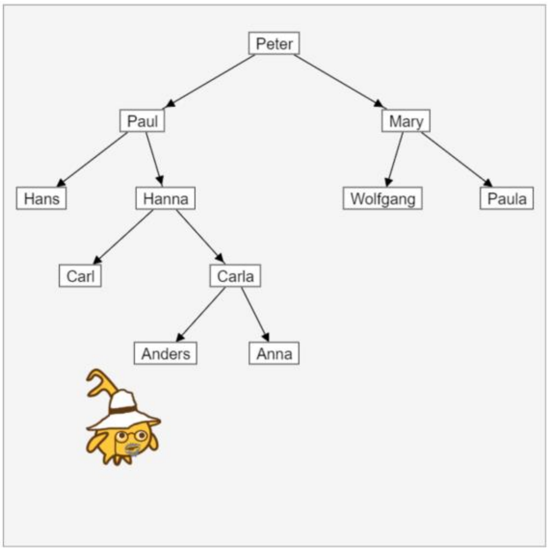
```
configure thisSprite as a GraphPad width: 400
height: 300 color: 245 245 245
repeat 300
    add 1 vertices to graph on thisSprite
    add 1 random edges to graph on thisSprite
configure DiagramSprite as a PlotPad width: 400
height: 300 color: 245 245 245
set PlotPad labels on DiagramSprite to
title: Edges per Node titleheight: 18
x-label: Number of Edges xLabelheight: 16
y-label: Number of Vertices yLabelheight: 16
add histogram of
    map number of vector keep items ( ≠ X) from ⌸ over
    adjacencyMatrix
with
10 groups
pretty formated? ✓ to PlotPad DiagramSprite
add axes and scales to PlotPad DiagramSprite
```

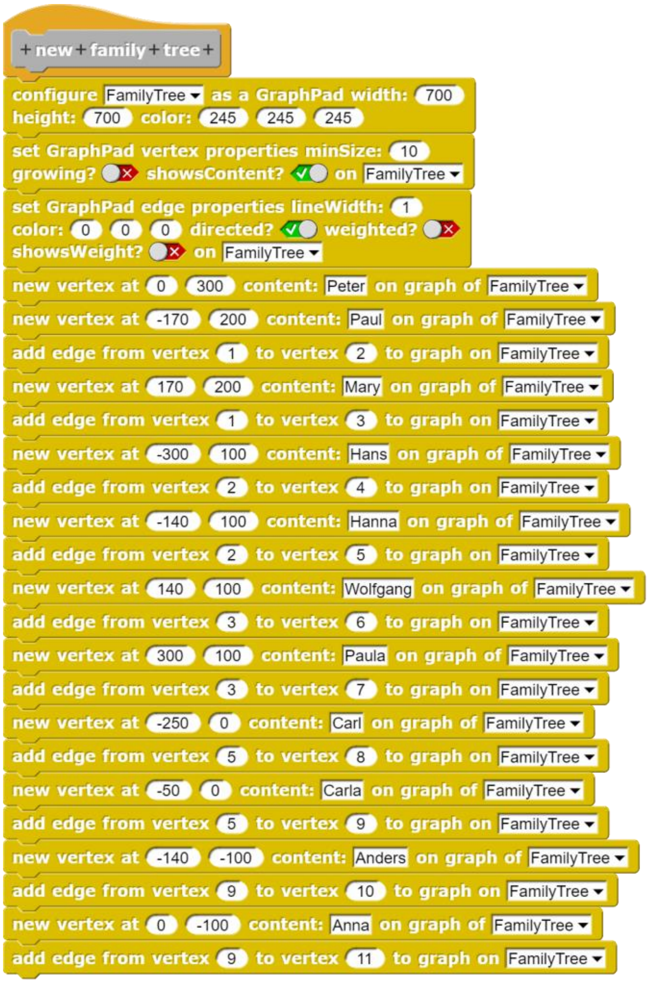




5.44 Breadth and depth search in the family tree

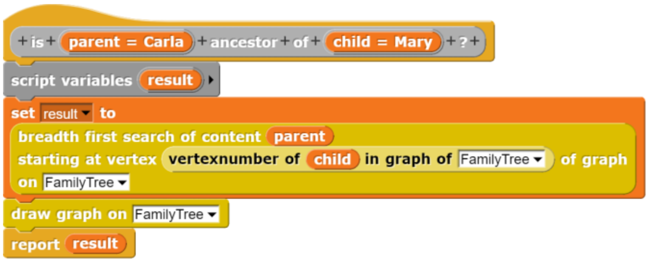
We create a family tree as a directed graph without edge weights simply by arranging and connecting the appropriate vertices. This is cumbersome, but simple. Just some fiddling.



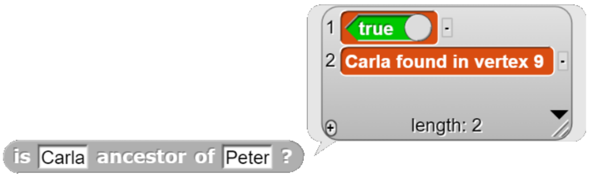


```
+new+family+tree+
configure FamilyTree as a GraphPad width: 700
height: 700 color: 245 245 245
set GraphPad vertex properties minSize: 10
growing? ☒ showsContent? ✓ on FamilyTree
set GraphPad edge properties lineWidth: 1
color: 0 0 0 directed? ✓ weighted? ☒
showsWeight? ☒ on FamilyTree
new vertex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

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



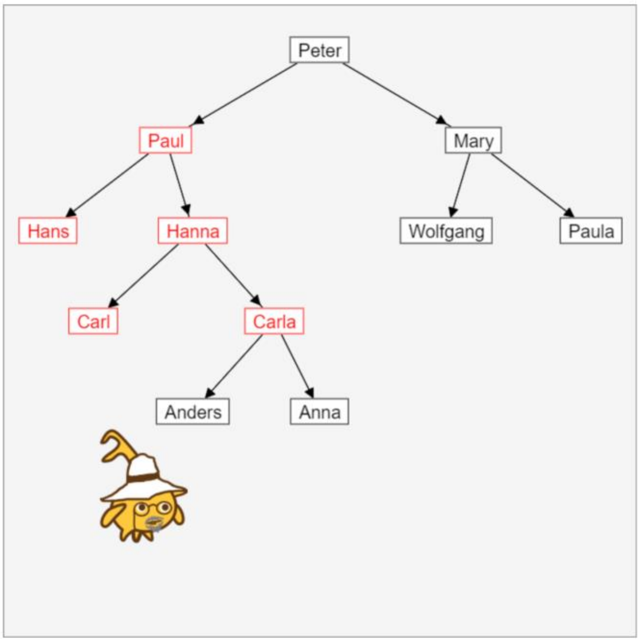
```
+is+ parent = Carla +ancestor+of+ child = Mary +?+
script variables result
set result to
  breadth first search of content parent
  starting at vertex vertexnumber of child in graph of FamilyTree of graph
  on FamilyTree
draw graph on FamilyTree
report result
```



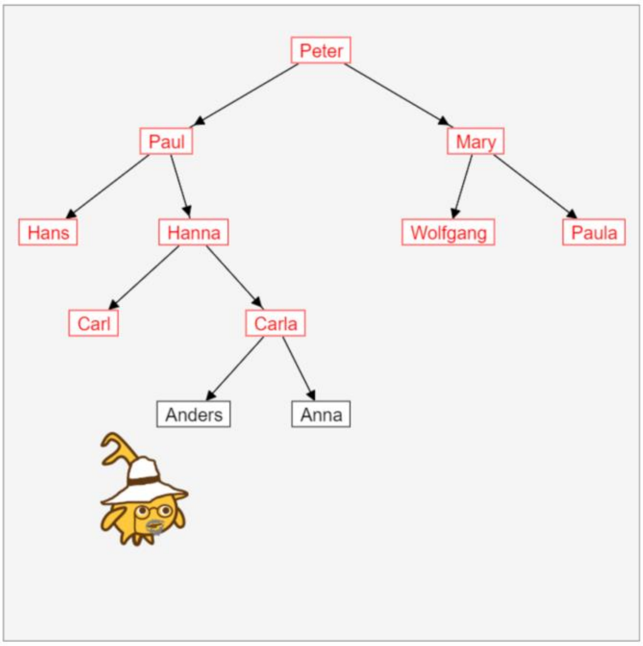
```
1 true
2 Carla found in vertex 9
length: 2
is Carla ancestor of Peter ?
```

In breadth 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.

depth search



breadth search



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 😉, …

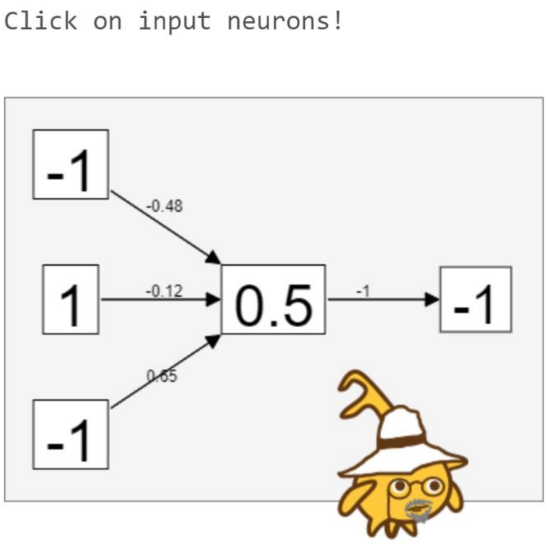
3. a: Develop a script for creating a decision tree, e.g. for classifying animals or plants: In each case, it is asked whether it is a particular specimen or, if not, what question can be used to distinguish the named one from the current one ("Does it have four legs?"). Either the specimen or the question is entered into the tree.

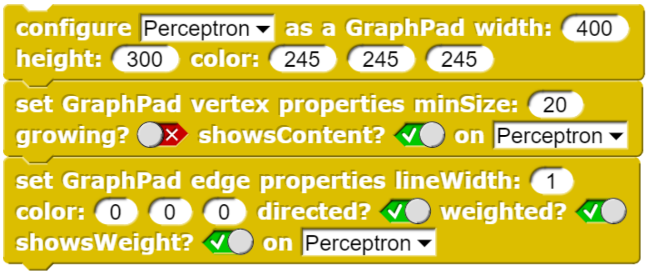
   b: Let others test your result. Try to estimate from what amount of data in the tree such a program would be useful to use.

   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

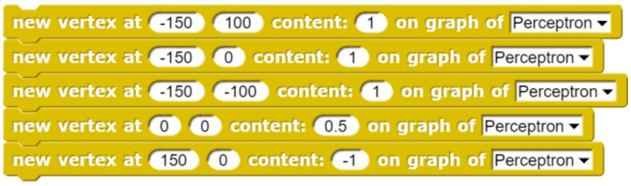
We want to ask *Hilberto* to use a *GraphPad* to illustrate how a simple *Perceptron* [22] works. To do this, he is to place three input nodes on the left of the image. in the middle sits the actual perceptron with a jump function that transmits either +1 or -1 to the output neuron on the right of the 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.





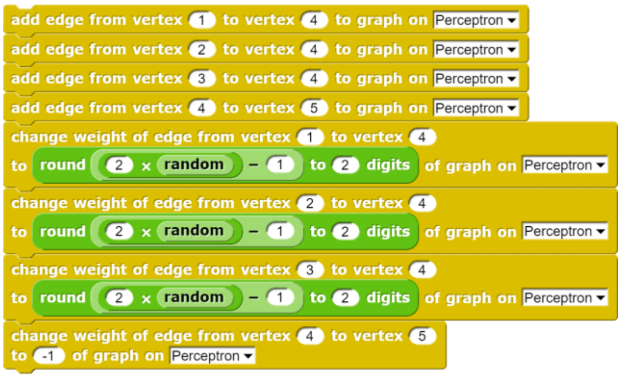
```
configure [Perceptron ▾] as a GraphPad width: (400)
height: (300) color: (245) (245) (245)
set GraphPad vertex properties minSize: (20)
growing? ☒ showsContent? ✓ on [Perceptron ▾]
set GraphPad edge properties lineWidth: (1)
color: (0) (0) (0) directed? ✓ weighted? ✓
showsWeight? ✓ on [Perceptron ▾]
```

Then he adds specified vertices of the perceptron net:



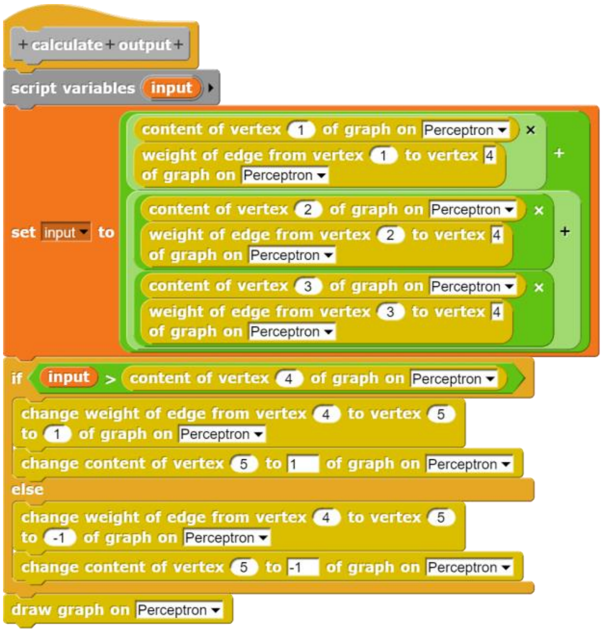
```
new vertex at (-150) (100) content: (1) on graph of [Perceptron ▾]
new vertex at (-150) (0) content: (1) on graph of [Perceptron ▾]
new vertex at (-150) (-100) content: (1) on graph of [Perceptron ▾]
new vertex at (0) (0) content: (0.5) on graph of [Perceptron ▾]
new vertex at (150) (0) content: (-1) on graph of [Perceptron ▾]
```

Only the edges are missing, first with random weights:



```
add edge from vertex (1) to vertex (4) to graph on [Perceptron ▾]
add edge from vertex (2) to vertex (4) to graph on [Perceptron ▾]
add edge from vertex (3) to vertex (4) to graph on [Perceptron ▾]
add edge from vertex (4) to vertex (5) to graph on [Perceptron ▾]
change weight of edge from vertex (1) to vertex (4)
to (round ((2) x (random)) - (1) to (2) digits) of graph on [Perceptron ▾]
change weight of edge from vertex (2) to vertex (4)
to (round ((2) x (random)) - (1) to (2) digits) of graph on [Perceptron ▾]
change weight of edge from vertex (3) to vertex (4)
to (round ((2) x (random)) - (1) to (2) digits) of graph on [Perceptron ▾]
change weight of edge from vertex (4) to vertex (5)
to (-1) of graph on [Perceptron ▾]
```

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.

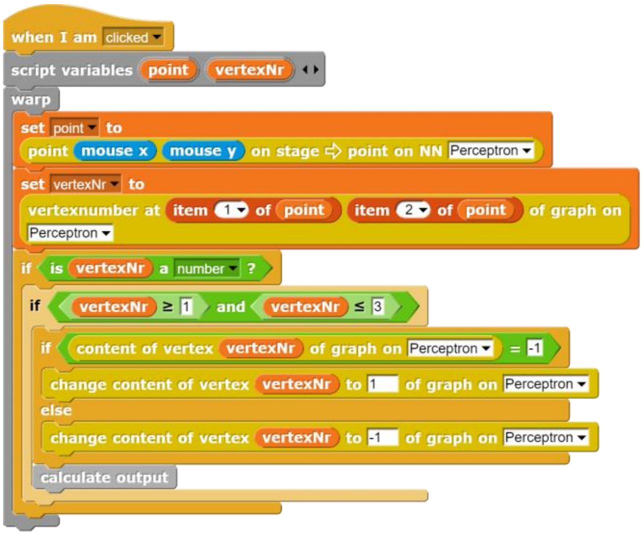


```
+calculate+output+
script variables (input) ▸
set [input ▾] to (((content of vertex (1) of graph on [Perceptron ▾]) × (weight of edge from vertex (1) to vertex [4] of graph on [Perceptron ▾]))
                 + (((content of vertex (2) of graph on [Perceptron ▾]) × (weight of edge from vertex (2) to vertex [4] of graph on [Perceptron ▾]))
                 + ((content of vertex (3) of graph on [Perceptron ▾]) × (weight of edge from vertex (3) to vertex [4] of graph on [Perceptron ▾]))))
if <(input) > (content of vertex (4) of graph on [Perceptron ▾])>
    change weight of edge from vertex (4) to vertex (5)
    to (1) of graph on [Perceptron ▾]
    change content of vertex (5) to [1] of graph on [Perceptron ▾]
else
    change weight of edge from vertex (4) to vertex (5)
    to (-1) of graph on [Perceptron ▾]
    change content of vertex (5) to [-1] of graph on [Perceptron ▾]
draw graph on [Perceptron ▾]
```

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

[22] 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.



```
when I am clicked
script variables point vertexNr
warp
  set point to point mouse x mouse y on stage ⇨ point on NN Perceptron
  set vertexNr to vertexnumber at item 1 of point item 2 of point of graph on Perceptron
  if is vertexNr a number ?
    if vertexNr ≥ 1 and vertexNr ≤ 3
      if content of vertex vertexNr of graph on Perceptron = -1
        change content of vertex vertexNr to 1 of graph on Perceptron
      else
        change content of vertex vertexNr to -1 of graph on Perceptron
      calculate output
```

Ready.

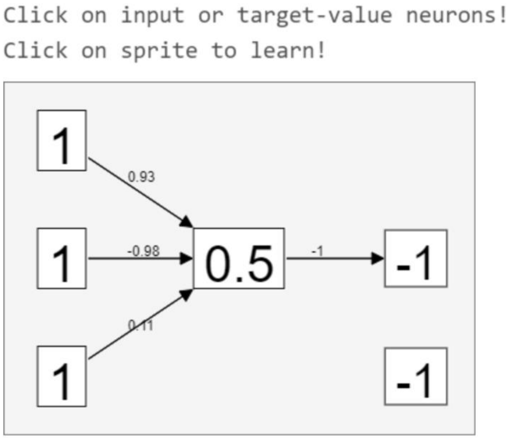
Hilberto himself is amazed that it can be done so easily. 😉

<u>Tasks</u>:

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

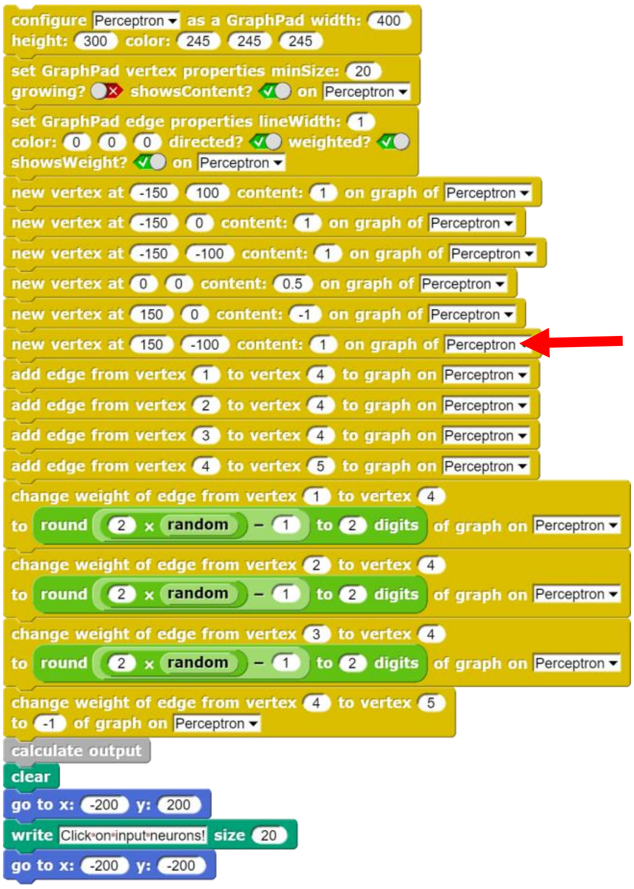
We now change the configuration a bit to teach the perceptron how to learn. To do this, we introduce another vertex, a "target vertex", under the output vertex, which displays "the correct" results, which in turn can be changed by clicking on it. If there is a difference between the values of the output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



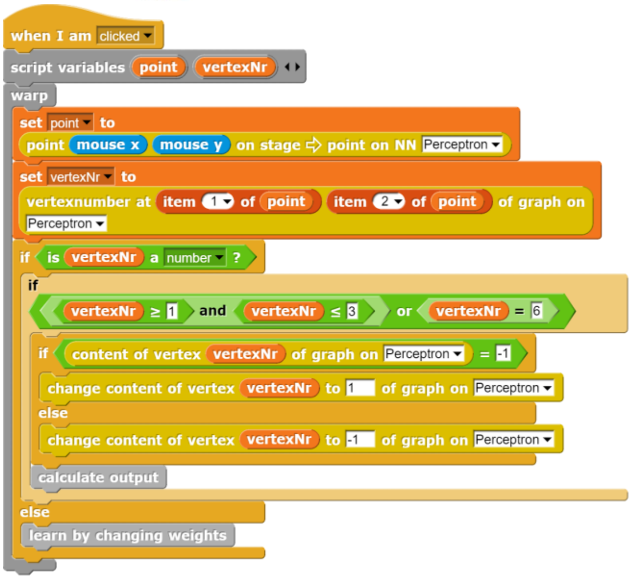
```
Click on input or target-value neurons!
Click on sprite to learn!
```

What to change?

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



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



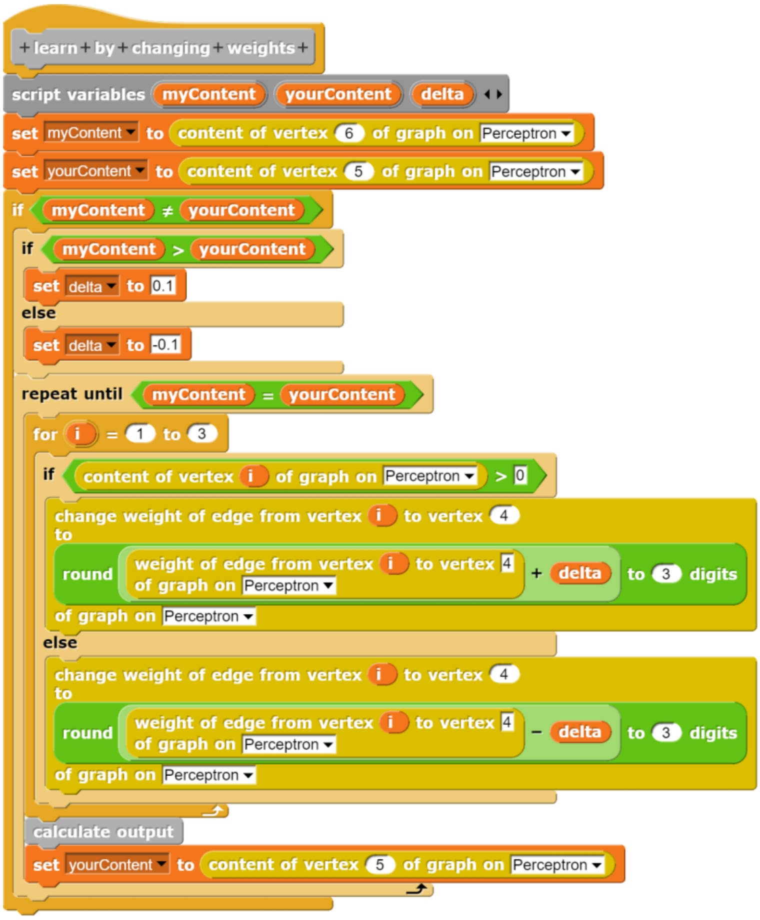
```
when I am clicked
script variables point vertexNr
warp
  set point to
    point mouse x mouse y on stage ⇨ point on NN Perceptron
  set vertexNr to
    vertexnumber at item 1 of point item 2 of point of graph on Perceptron
  if is vertexNr a number ?
    if <<vertexNr ≥ 1> and <vertexNr ≤ 3>> or <vertexNr = 6>
      if content of vertex vertexNr of graph on Perceptron = -1
        change content of vertex vertexNr to 1 of graph on Perceptron
      else
        change content of vertex vertexNr to -1 of graph on Perceptron
      calculate output
  else
    learn by changing weights
```

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.



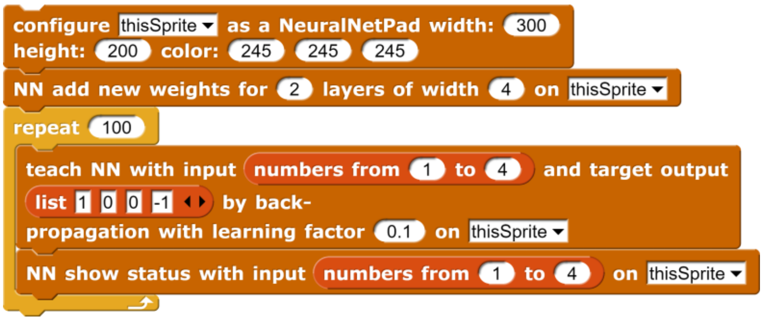
```
+learn+by+changing+weights+
script variables myContent yourContent delta
set myContent to content of vertex 6 of graph on Perceptron
set yourContent to content of vertex 5 of graph on Perceptron
if myContent ≠ yourContent
  if myContent > yourContent
    set delta to 0.1
  else
    set delta to -0.1
  repeat until myContent = yourContent
    for i = 1 to 3
      if content of vertex i of graph on Perceptron > 0
        change weight of edge from vertex i to vertex 4
        to
          round weight of edge from vertex i to vertex 4
                of graph on Perceptron + delta to 3 digits
        of graph on Perceptron
      else
        change weight of edge from vertex i to vertex 4
        to
          round weight of edge from vertex i to vertex 4
                of graph on Perceptron − delta to 3 digits
        of graph on Perceptron
    calculate output
    set yourContent to content of vertex 5 of graph on Perceptron
```

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!

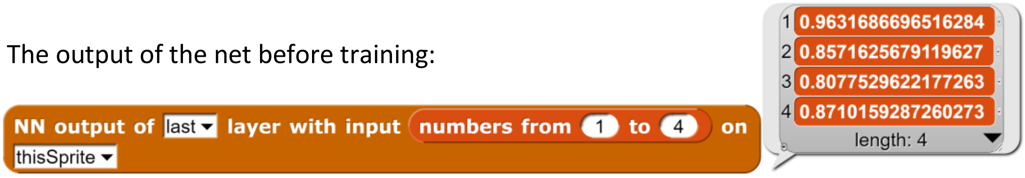
5.47 Training of a neural network

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.



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

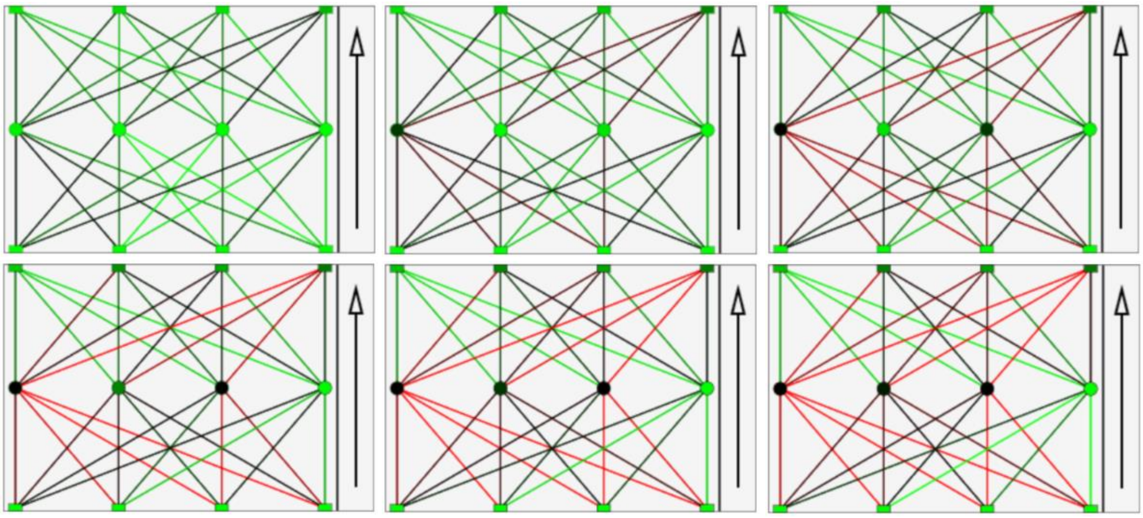
The output of the net before training:



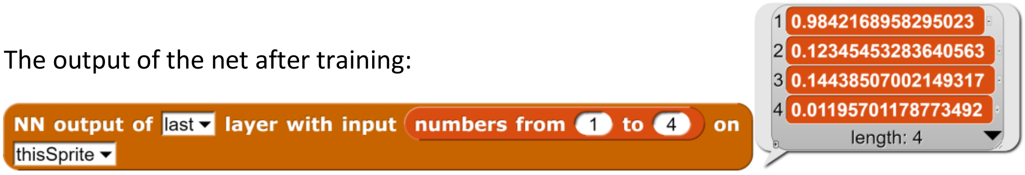
```
NN output of last layer with input (numbers from 1 to 4) on
thisSprite
```

```
1 0.9631686696516284
2 0.8571625679119627
3 0.8077529622177263
4 0.8710159287260273
length: 4
```

Training states after every 20 training steps:



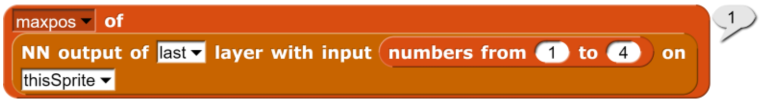
The output of the net after training:



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

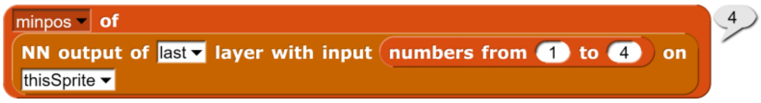
```
1 0.9842168958295023
2 0.12345453283640563
3 0.14438507002149317
4 0.011957011787773492
length: 4
```

The positions of the largest or smallest value of the output can be determined directly using the *SciSnap!DataLibrary*:



```
maxpos of
    NN output of last layer with input (numbers from 1 to 4) on
    thisSprite
```

```
1
```



```
minpos of
    NN output of last layer with input (numbers from 1 to 4) on
    thisSprite
```

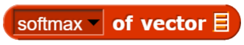
```
4
```

5.48 Traffic sign recognition with a neural network of perceptrons

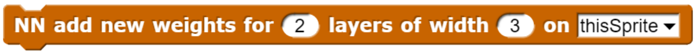
"Deep" neural networks dominate the discussion about current "artificial intelligence". These are mostly "fully connected" networks consisting of several perceptron layers. "Fully connected" means that all neurons in one layer are connected to all of the next layer. Each connection is assigned a weight 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.

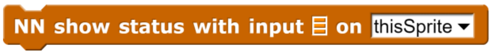
One way to train perceptron networks is to present them input vectors and the desired output at the same time. The network then computes the output resulting from the available, initially randomly chosen weights, and determines the difference to the given result. Starting from the last layer of results, 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  softmax of vector which can be used to scale input vectors, for example. You should also inform yourself about this.

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

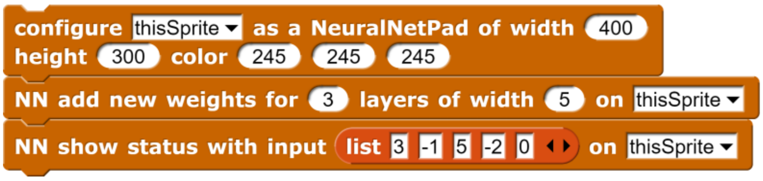


NN add new weights for 2 layers of width 3 on thisSprite



NN show status with input on thisSprite

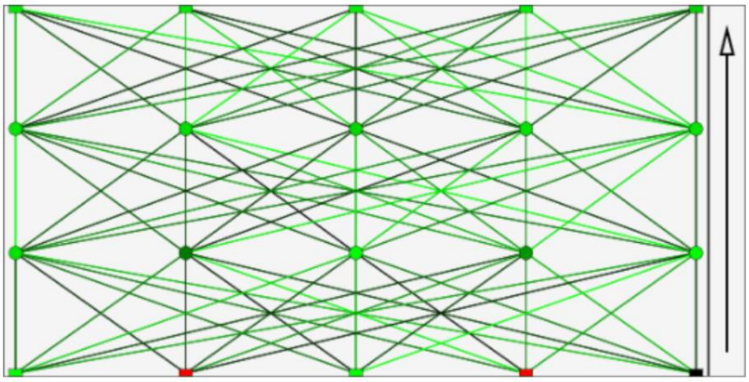
Since displaying the many numbers would be rather confusing and also hardly informative, the connecting lines (the edges) are color-coded according to the values of the associated weights: from full green for large positive values to black for small amounts to red negative weights. Since initially only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



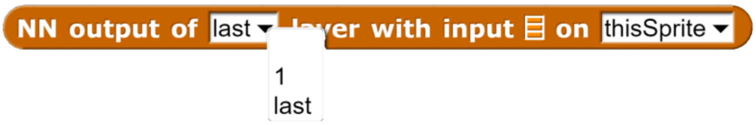
configure thisSprite as a NeuralNetPad of width 400 height 300 color 245 245 245

NN add new weights for 3 layers of width 5 on thisSprite

NN show status with input list 3 -1 5 -2 0 on thisSprite

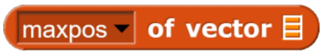


The vertices of the net are color coded like the edges. At the bottom, the elements of the input vector are shown as small rectangles. The inner layers form colored circles and the last layer is again shown as an output layer in rectangular form. The direction of the calculation from bottom to top is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



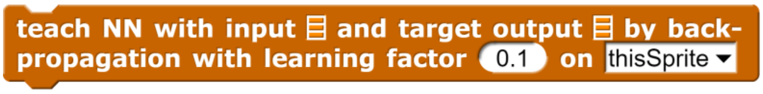
NN output of last layer with input ⌸ on thisSprite

1
last



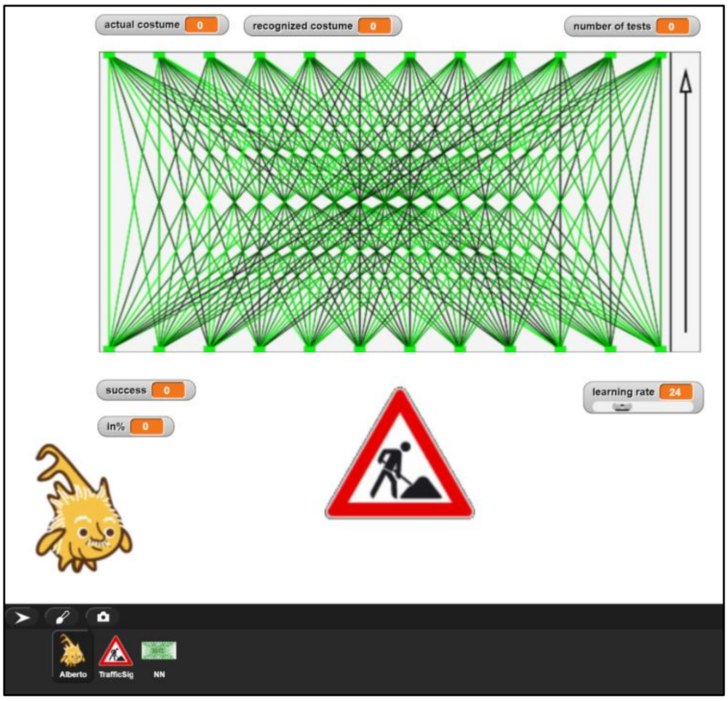
maxpos of vector ⌸

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..



teach NN with input ⌸ and target output ⌸ by backpropagation with learning factor 0.1 on thisSprite

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.



actual costume 0

recognized costume 0

number of tests 0

success 0

in% 0

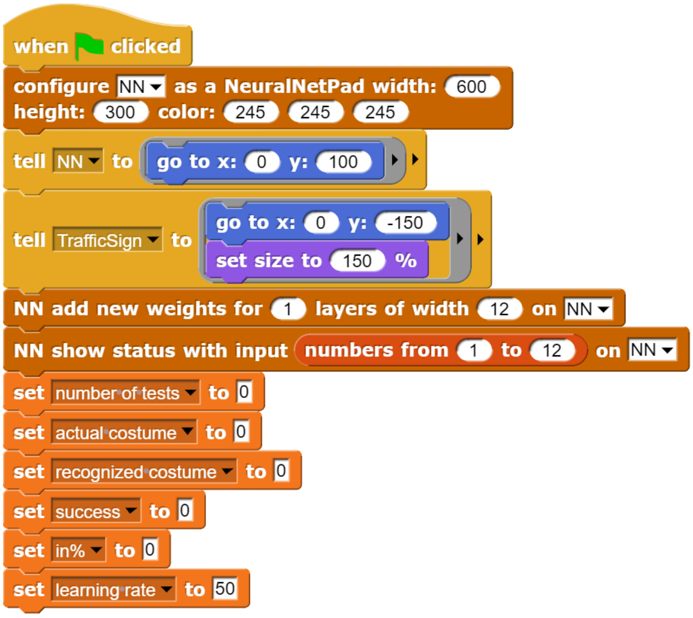
learning rate 24

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.

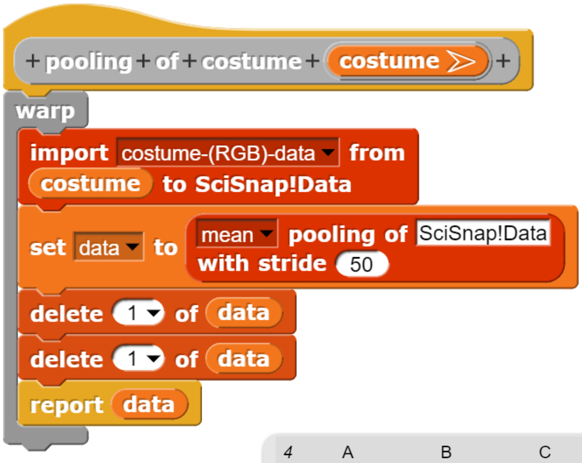
Because it is a difficult problem, this time *Alberto* takes the overall control.

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.

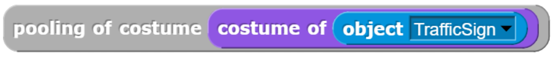


```
when ⚑ clicked
configure NN as a NeuralNetPad width: 600
height: 300 color: 245 245 245
tell NN to (go to x: 0 y: 100)
tell TrafficSign to (go to x: 0 y: -150
                     set size to 150 %)
NN add new weights for 1 layers of width 12 on NN
NN show status with input (numbers from 1 to 12) on NN
set number of tests to 0
set actual costume to 0
set recognized costume to 0
set success to 0
set in% to 0
set learning rate to 50
```

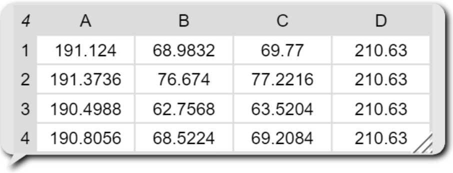
Alberto first needs to reduce the amount of image values using the *pooling* operation. To apply the operation, it must import the image data. Then it can convert them, delete the dimensions of the reduced image specified at the front of the list, and return the result. We summarize everything in a new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



```
+ pooling + of + costume + (costume >>) +
warp
  import costume-(RGB)-data from
  costume to SciSnap!Data
  set data to (mean pooling of SciSnap!Data
               with stride 50)
  delete 1 of data
  delete 1 of data
  report data
```

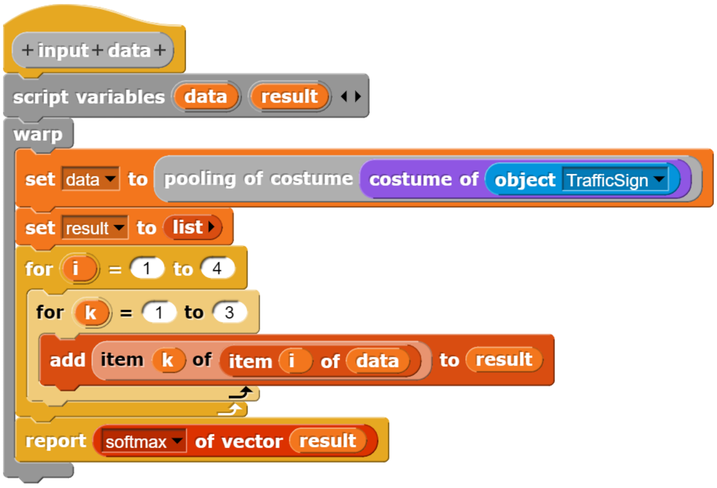


```
pooling of costume (costume of (object TrafficSign))
```



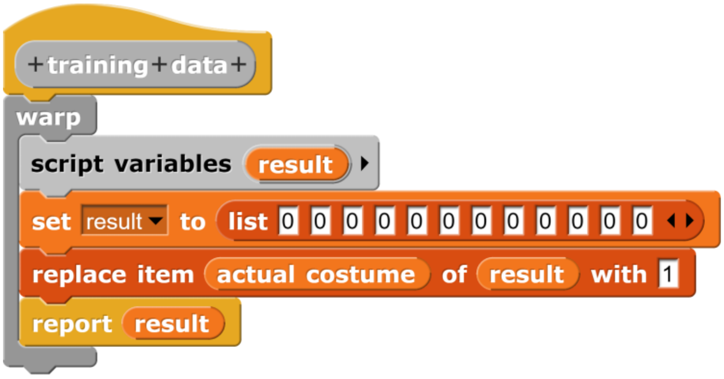
4	A	B	C	D
1	191.124	68.9832	69.77	210.63
2	191.3736	76.674	77.2216	210.63
3	190.4988	62.7568	63.5204	210.63
4	190.8056	68.5224	69.2084	210.63

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.



```
+ input + data +
script variables data result
warp
  set data to (pooling of costume (costume of (object TrafficSign)))
  set result to (list)
  for i = 1 to 4
    for k = 1 to 3
      add (item k of (item i of data)) to result
  report (softmax of vector result)
```

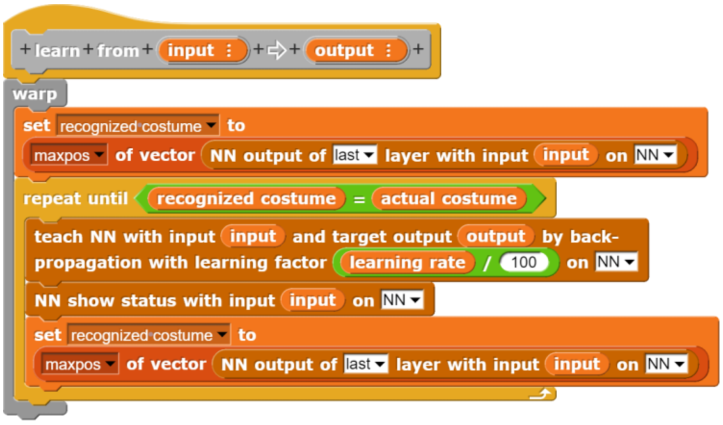
Accordingly, the training vector in *training data* can be determined with the searched output values of the NN. In our case, all values should be 0 except for the one corresponding to the costume number of the traffic sign.



```
+training+data+
warp
  script variables (result)
  set result to list 0 0 0 0 0 0 0 0 0 0 0 0
  replace item (actual costume) of (result) with 1
  report (result)
```

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.

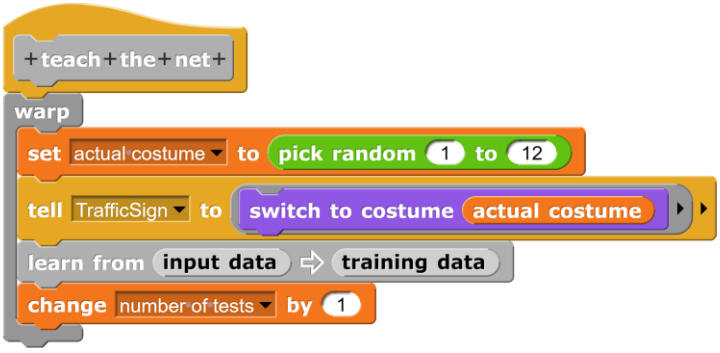
For testing, the same operation is performed only once.



```
+learn+from+ (input :) +⇨+ (output :) +
warp
  set recognized costume to
    maxpos of vector (NN output of last layer with input (input) on NN)
  repeat until <(recognized costume) = (actual costume)>
    teach NN with input (input) and target output (output) by back-
    propagation with learning factor ((learning rate) / 100) on NN
    NN show status with input (input) on NN
    set recognized costume to
      maxpos of vector (NN output of last layer with input (input) on NN)
```

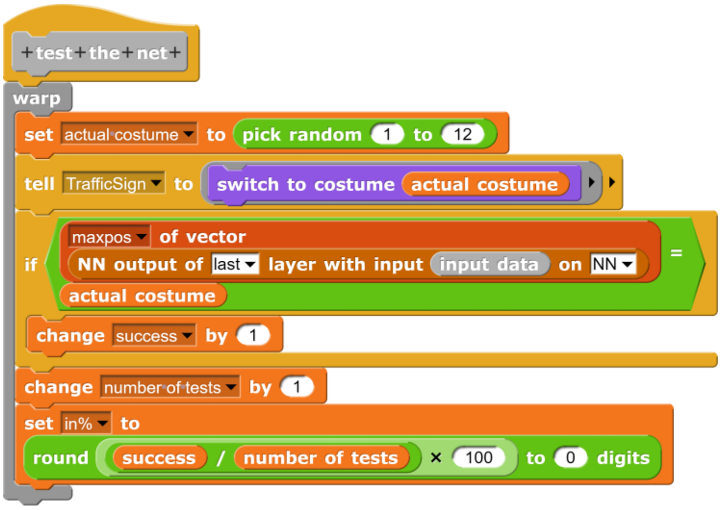
Now we have everything together to let *Alberto* work reasonably.

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.



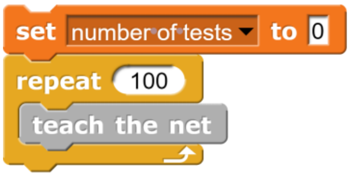
```
+teach+the+net+
warp
  set actual costume to (pick random 1 to 12)
  tell TrafficSign to (switch to costume (actual costume))
  learn from (input data) ⇨ (training data)
  change number of tests by 1
```

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.

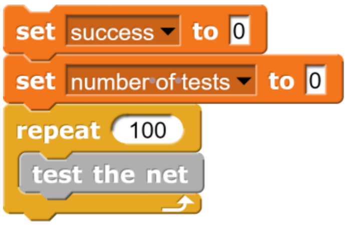


```
+test+the+net+
warp
  set actual costume to (pick random 1 to 12)
  tell TrafficSign to (switch to costume (actual costume))
  if <(maxpos of vector (NN output of last layer with input (input data) on NN)) = (actual costume)>
    change success by 1
  change number of tests by 1
  set in% to
    round (((success) / (number of tests)) × 100) to 0 digits
```

Multiple learning and test runs can then be easily triggered.

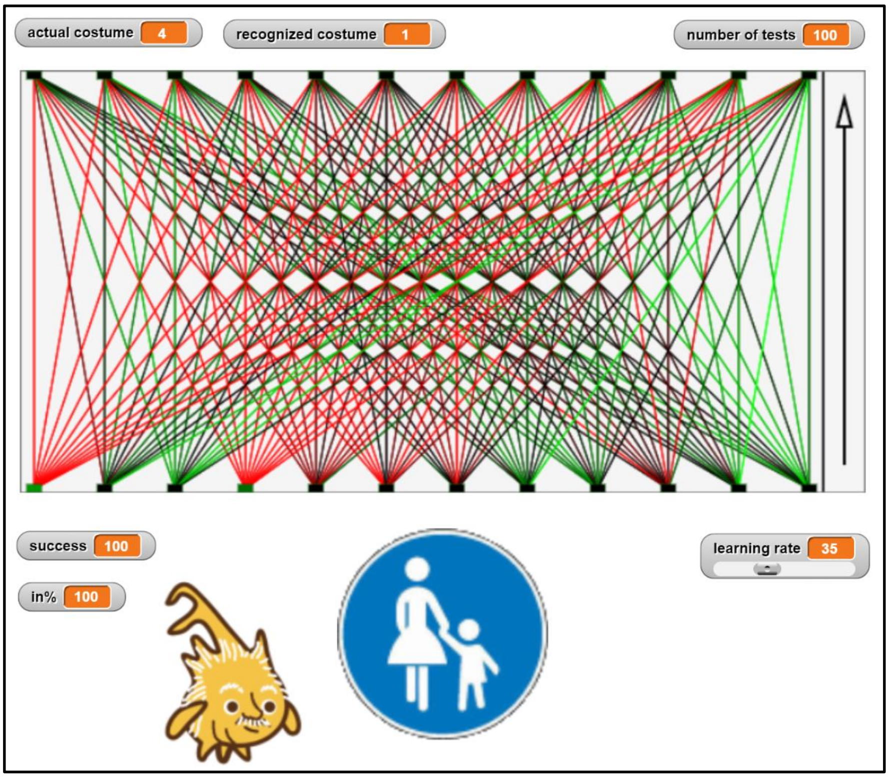


```
set number of tests to 0
repeat 100
  teach the net
```



```
set success to 0
set number of tests to 0
repeat 100
  test the net
```

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%.



actual costume 4

recognized costume 1

number of tests 100

success 100

in% 100

learning rate 35

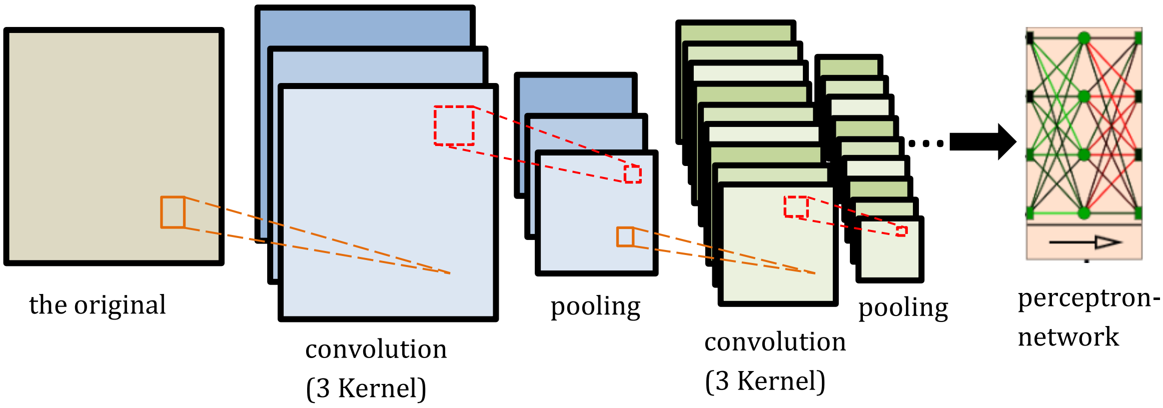
Tasks:

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

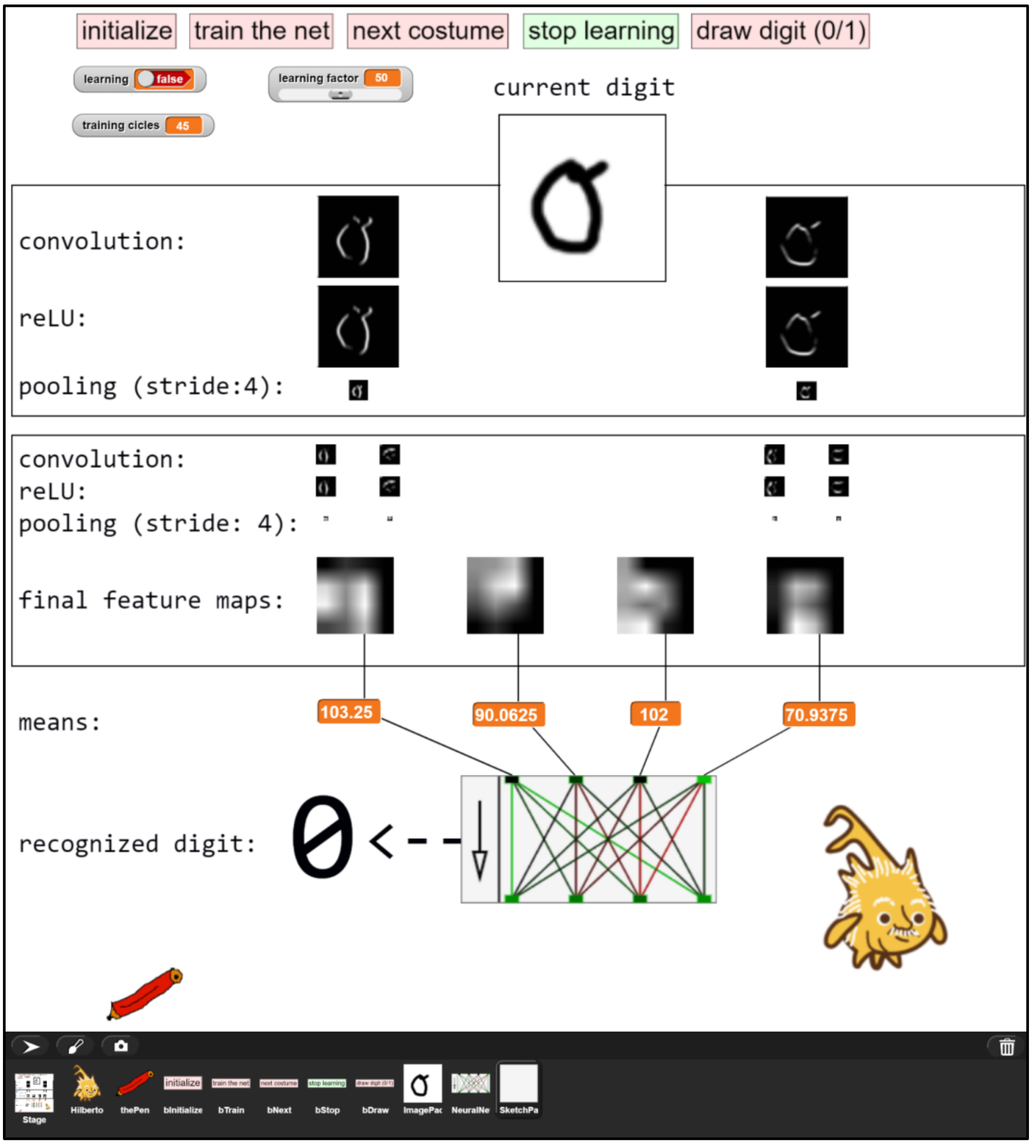
The immense number of parameters in fully connected perceptron networks and the consequent need for huge amounts of training data has led to other network variants to drastically reduce this number. One of these is *Convolutional Neural Networks (CNNs*), in which the amount of input data to the perceptron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



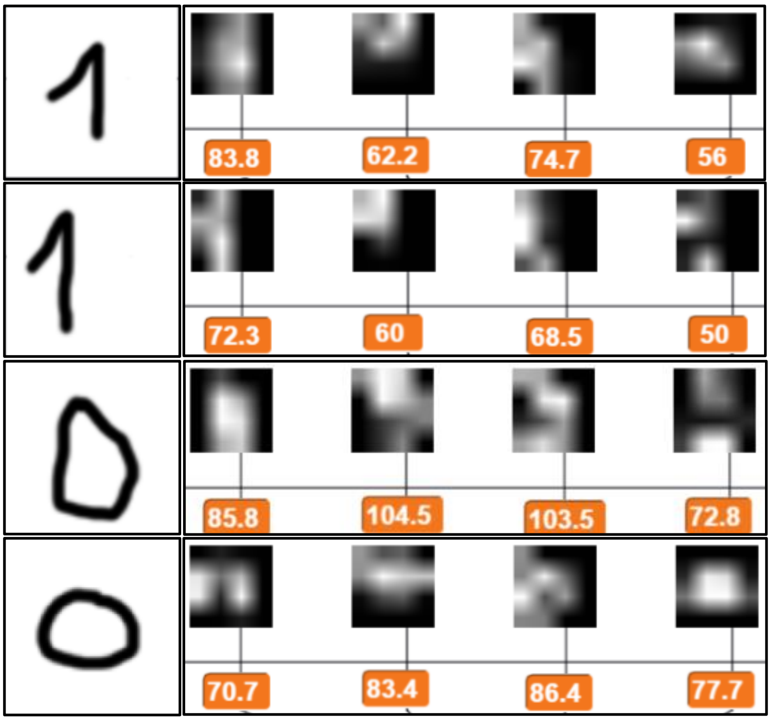
As an example, let's take a kernel that filters vertical lines: it colors a point white if there is a second pixel next to the point, otherwise black. In the "folded" image we can then recognize vertical lines of the original as bright spots. If it does not matter so much where exactly these lines are, 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.



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.

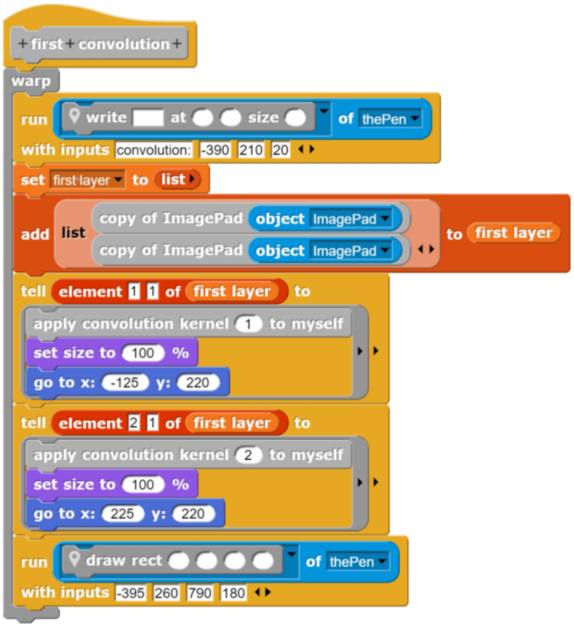
In the image above, after two stages of reduction, four feature maps of 16x16 pixels each are left, each of which has been run twice through the operations Convolution reLU Max-Pooling: on the far left with the kernel for vertical lines, then with both kernels in different order, and finally twice with 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.



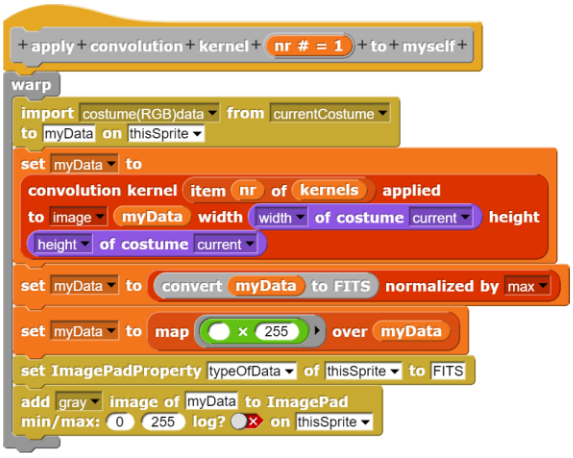
In the image above, after two stages of reduction, four feature maps of 16x16 pixels each are left, each of which has been run twice through the operations *Convolution → reLU → Max-Pooling*: on the far left with the kernel for vertical lines, then with both kernels in different order, and finally twice 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.

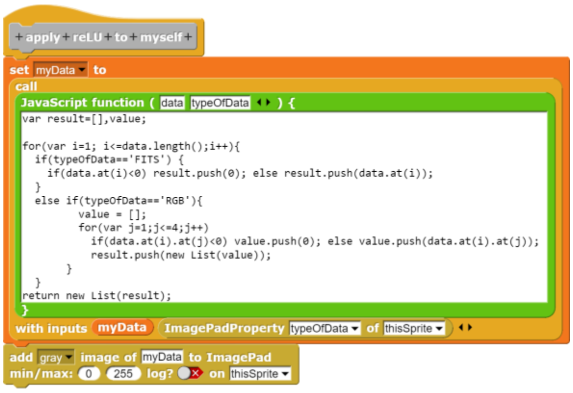


```
+first+convolution+
warp
  run (write [] at () () size () of thePen) with inputs [convolution:] [-390] [210] [20]
  set first layer to list
  add list (copy of ImagePad object ImagePad) (copy of ImagePad object ImagePad) to first layer
  tell element [1] [1] of first layer to
    apply convolution kernel (1) to myself
    set size to (100) %
    go to x: (-125) y: (220)
  tell element [2] [1] of first layer to
    apply convolution kernel (2) to myself
    set size to (100) %
    go to x: (225) y: (220)
  run (draw rect () () () () of thePen) with inputs [-395] [260] [790] [180]
```



```
+apply+convolution+kernel+(nr # = 1)+to+myself+
warp
  import costume(RGB)data from currentCostume to [myData] on thisSprite
  set myData to convolution kernel (item (nr) of (kernels)) applied to image (myData) width (width of costume current) height (height of costume current)
  set myData to (convert (myData) to FITS) normalized by max
  set myData to map (( ) × (255)) over (myData)
  set ImagePadProperty typeOfData of thisSprite to [FITS]
  add gray image of [myData] to ImagePad min/max: (0) (255) log? ☒ on thisSprite
```

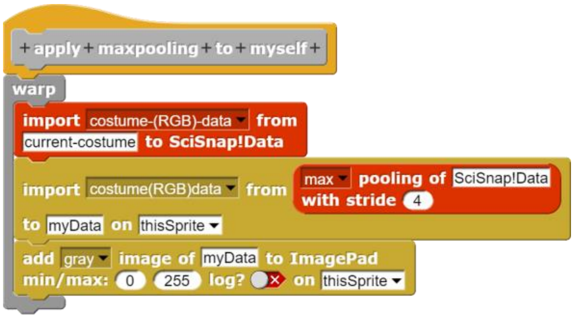
After that, each copy must pass through a *reLU (rectified linear unit)*, which serves as an activation function. In this case, negative values are simply set to zero.



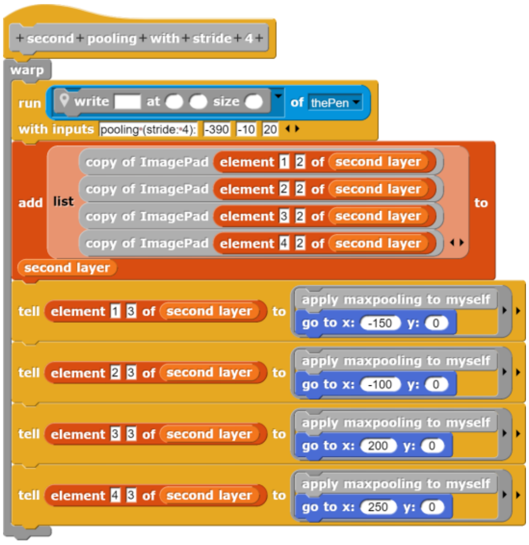
```
+apply+reLU+to+myself+
set myData to
  call
    JavaScript function ( [data] [typeOfData] ) {
var result=[],value;

for(var i=1; i<=data.length();i++){
  if(typeOfData=='FITS') {
    if(data.at(i)<0) result.push(0); else result.push(data.at(i));
  }
  else if(typeOfData=='RGB'){
         value = [];
         for(var j=1;j<=4;j++)
           if(data.at(i).at(j)<0) value.push(0); else value.push(data.at(i).at(j));
           result.push(new List(value));
       }
  }
return new List(result);
}
  with inputs (myData) (ImagePadProperty typeOfData of thisSprite)
add gray image of [myData] to ImagePad min/max: (0) (255) log? ☒ on thisSprite
```

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.

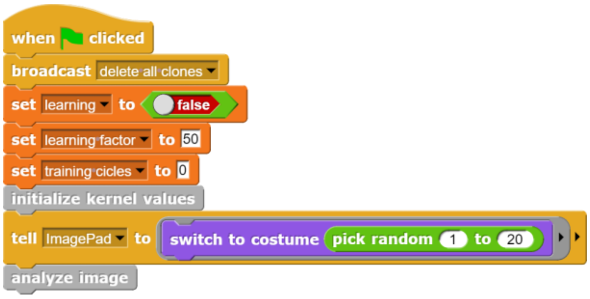


```
+apply+maxpooling+to+myself+
warp
  import costume-(RGB)-data from current-costume to SciSnap!Data
  import costume(RGB)data from (max pooling of SciSnap!Data with stride 4) to myData on thisSprite
  add gray image of myData to ImagePad min/max: 0 255 log? ○ on thisSprite
```

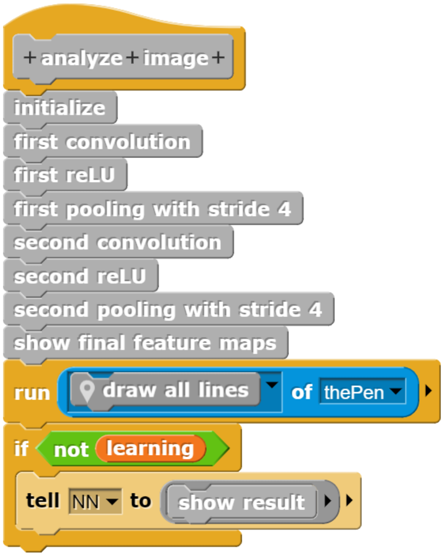


```
+second+pooling+with+stride+4+
warp
  run (write [] at () () size ()) of thePen with inputs pooling·(stride:·4): -390 -10 20
  add (list (copy of ImagePad (element 1 2 of (second layer)))
            (copy of ImagePad (element 2 2 of (second layer)))
            (copy of ImagePad (element 3 2 of (second layer)))
            (copy of ImagePad (element 4 2 of (second layer)))) to (second layer)
  tell (element 1 3 of (second layer)) to (apply maxpooling to myself
                                           go to x: -150 y: 0)
  tell (element 2 3 of (second layer)) to (apply maxpooling to myself
                                           go to x: -100 y: 0)
  tell (element 3 3 of (second layer)) to (apply maxpooling to myself
                                           go to x: 200 y: 0)
  tell (element 4 3 of (second layer)) to (apply maxpooling to myself
                                           go to x: 250 y: 0)
```

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.



```
when ⚑ clicked
broadcast delete all clones
set learning to false
set learning factor to 50
set training cicles to 0
initialize kernel values
tell ImagePad to (switch to costume (pick random 1 to 20))
analyze image
```

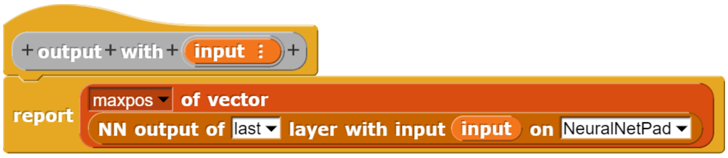


```
+analyze+image+
initialize
first convolution
first reLU
first pooling with stride 4
second convolution
second reLU
second pooling with stride 4
show final feature maps
run (draw all lines) of thePen
if not learning
  tell NN to (show result)
```

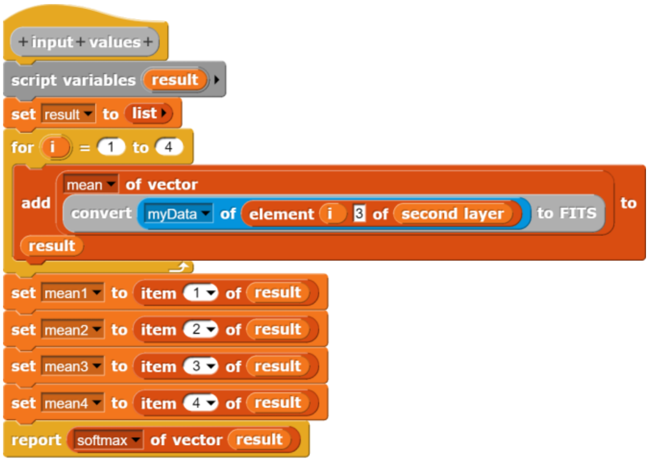
The *initialize* method only takes care of drawing the lines on the stage. The other methods work with two layers of the CNN, *first layer* and *second layer*, each containing the versions of the digits that appear on the stage. So that they do not interfere with each other, they work with copies of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

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

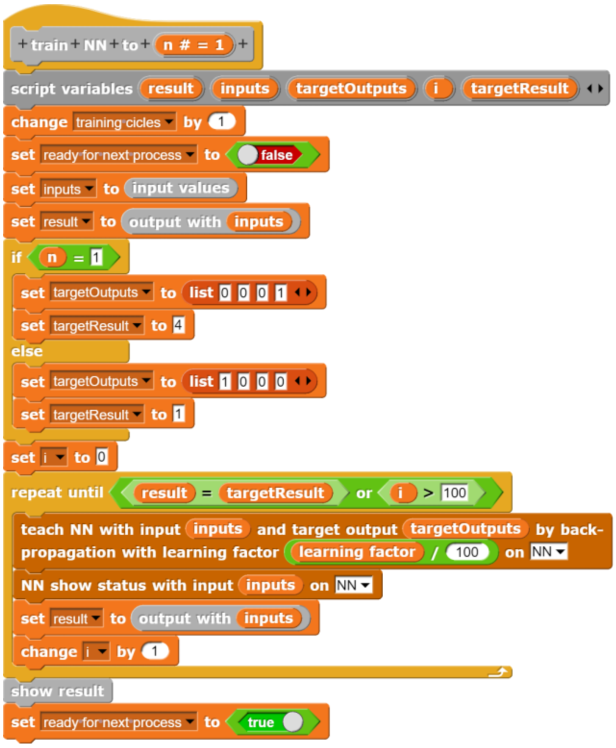
The neural network in the form of a *NeuralNetPad* is supposed to generate the largest output at output *1* for zeros and at output *4* for ones. This is of course completely arbitrary. The current output value is determined by the function *output with <input>*. With its components the net can be trained,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



```
+output+with+input :+
report (maxpos of vector (NN output of last layer with input (input) on NeuralNetPad))
```

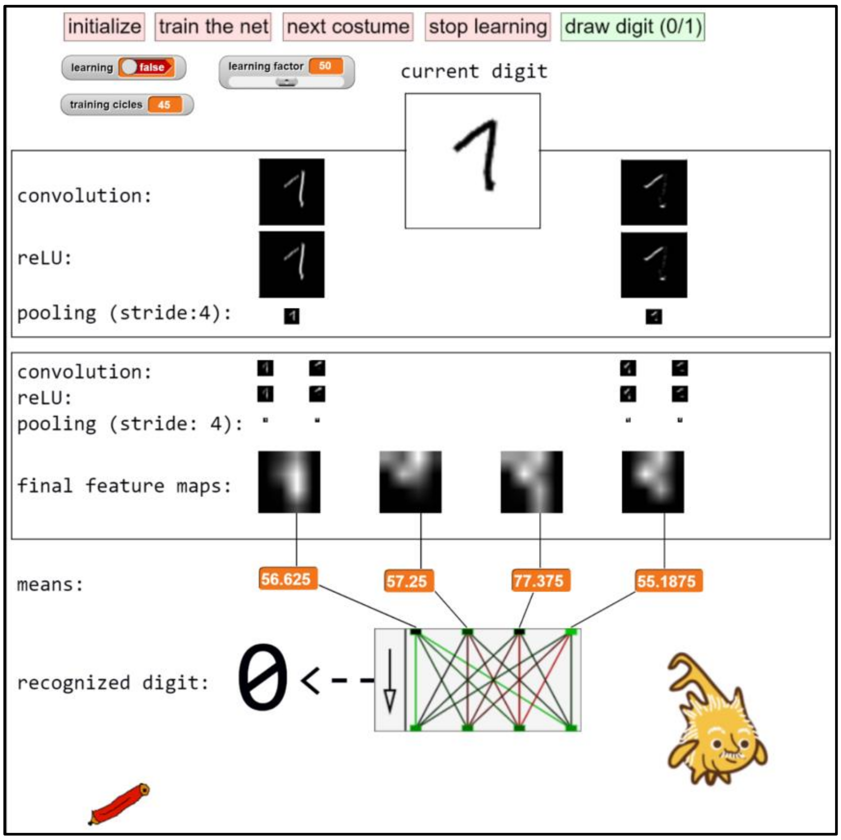


```
+input+values+
script variables (result)
set result to list
for i = 1 to 4
    add (mean of vector (convert (myData of element i 3 of second layer) to FITS)) to result
set mean1 to item 1 of result
set mean2 to item 2 of result
set mean3 to item 3 of result
set mean4 to item 4 of result
report softmax of vector result
```



```
+train+NN+to+ (n # = 1) +
script variables (result) (inputs) (targetOutputs) (i) (targetResult)
change training cicles by 1
set ready for next process to false
set inputs to input values
set result to output with inputs
if n = 1
    set targetOutputs to list 0 0 0 1
    set targetResult to 4
else
    set targetOutputs to list 1 0 0 0
    set targetResult to 1
set i to 0
repeat until <(result = targetResult) or (i > 100)>
    teach NN with input inputs and target output targetOutputs by back-
    propagation with learning factor (learning factor / 100) on NN
    NN show status with input inputs on NN
    set result to output with inputs
    change i by 1
show result
set ready for next process to true
```

And - has the network learned anything? Let's write a number:



learning false

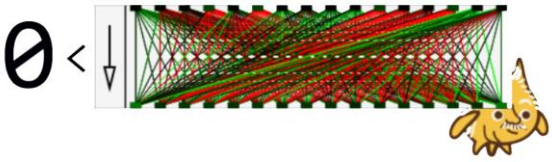
learning factor 50

training cicles 45

Well - there is still room for improvement!

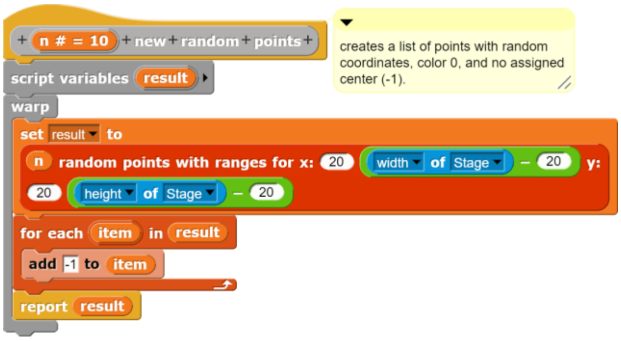
Tasks:

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.



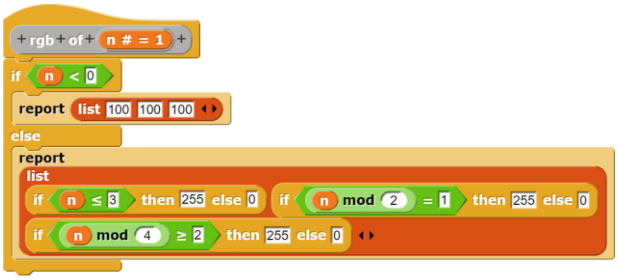
5.51 k-means-Clustering

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.

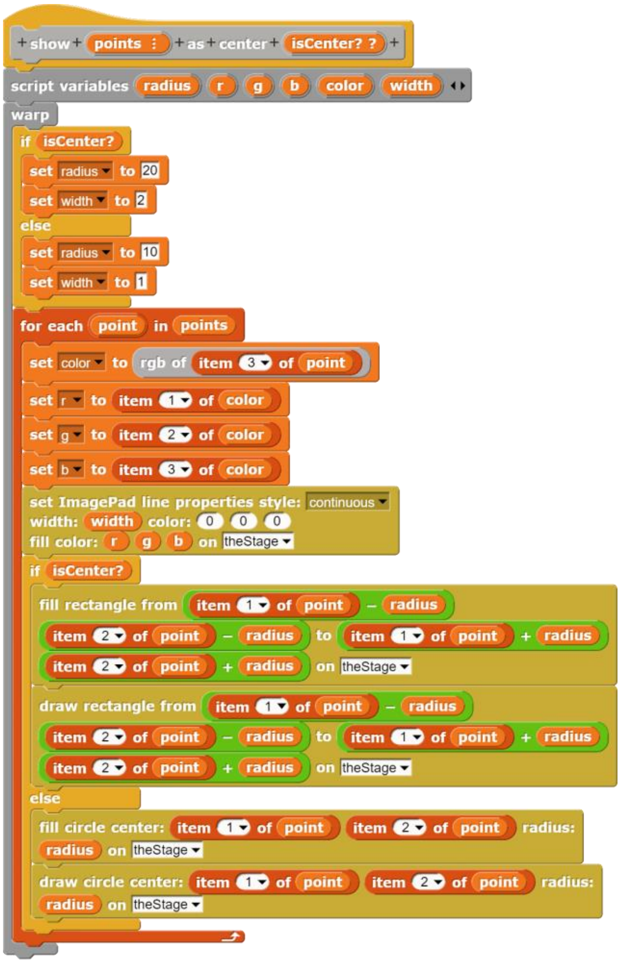


```
+ n # = 10 + new + random + points +
script variables result
warp
    set result to
        n random points with ranges for x: 20 (width of Stage) − 20 y:
        20 (height of Stage) − 20
    for each item in result
        add -1 to item
    report result
```

creates a list of points with random coordinates, color 0, and no assigned center (-1).

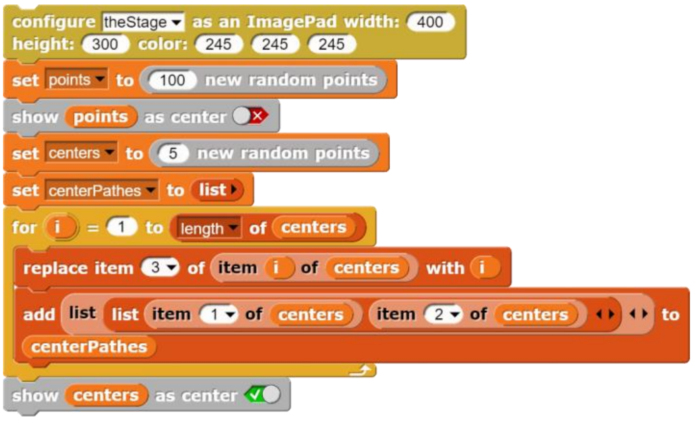


```
+ rgb + of + n # = 1 +
if n < 0
    report list 100 100 100
else
    report
        list
            if n ≤ 3 then 255 else 0    if (n mod 2) = 1 then 255 else 0
            if (n mod 4) ≥ 2 then 255 else 0
```



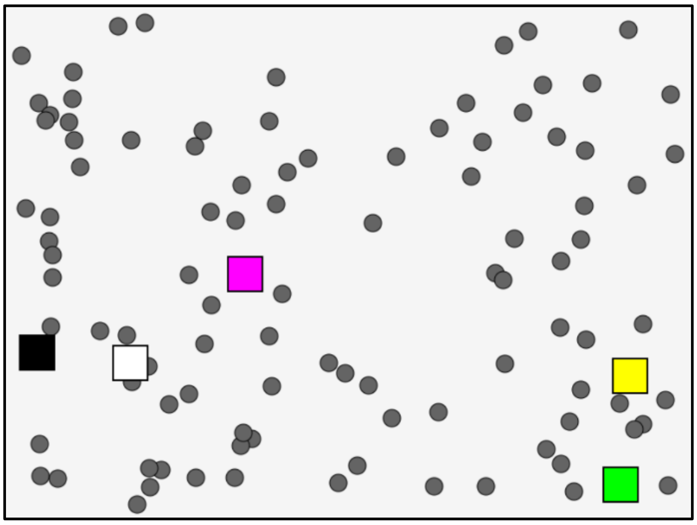
```
+ show + points : + as + center + isCenter? ? +
script variables radius r g b color width
warp
    if isCenter?
        set radius to 20
        set width to 2
    else
        set radius to 10
        set width to 1
    for each point in points
        set color to rgb of item 3 of point
        set r to item 1 of color
        set g to item 2 of color
        set b to item 3 of color
        set ImagePad line properties style: continuous
        width: width color: 0 0 0
        fill color: r g b on theStage
        if isCenter?
            fill rectangle from (item 1 of point) − radius
            (item 2 of point) − radius to (item 1 of point) + radius
            (item 2 of point) + radius on theStage
            draw rectangle from (item 1 of point) − radius
            (item 2 of point) − radius to (item 1 of point) + radius
            (item 2 of point) + radius on theStage
        else
            fill circle center: item 1 of point item 2 of point radius:
            radius on theStage
            draw circle center: item 1 of point item 2 of point radius:
            radius on theStage
```

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

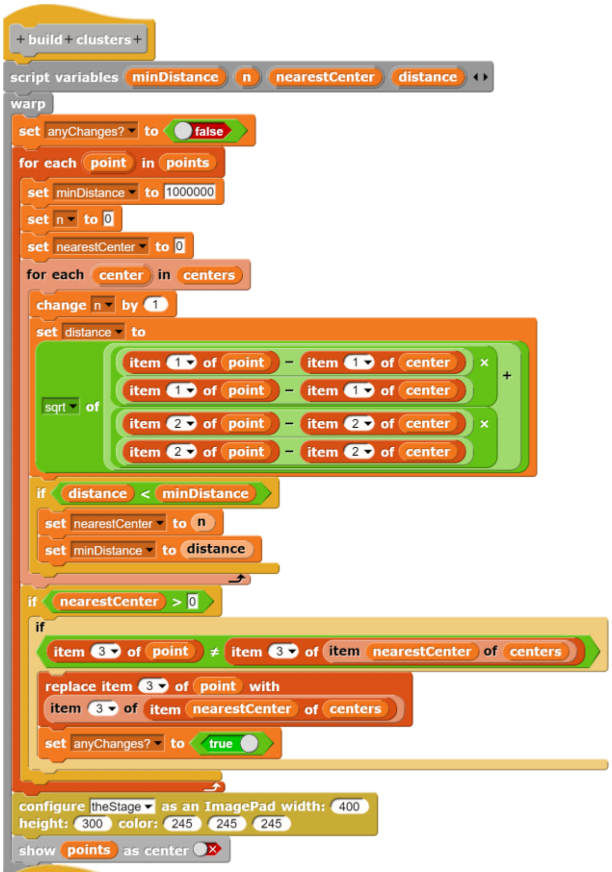


```
configure theStage as an ImagePad width: 400
height: 300 color: 245 245 245
set points to 100 new random points
show points as center ☒
set centers to 5 new random points
set centerPathes to list
for i = 1 to length of centers
    replace item 3 of item i of centers with i
    add list list item 1 of centers item 2 of centers to
    centerPathes
show centers as center ✓
```

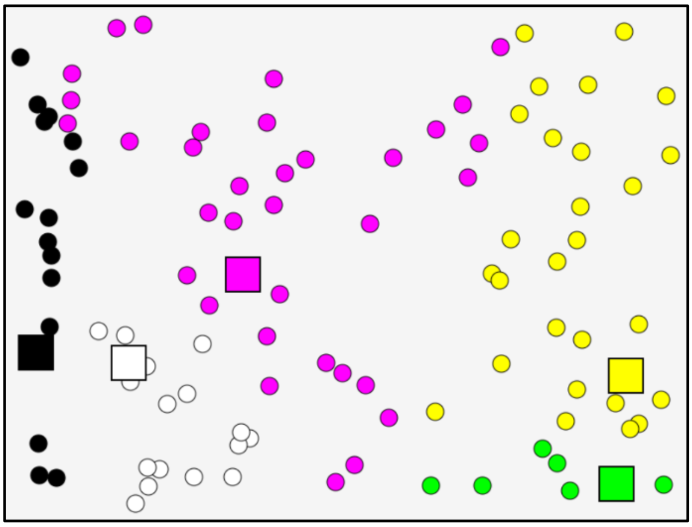
... and get the following image:



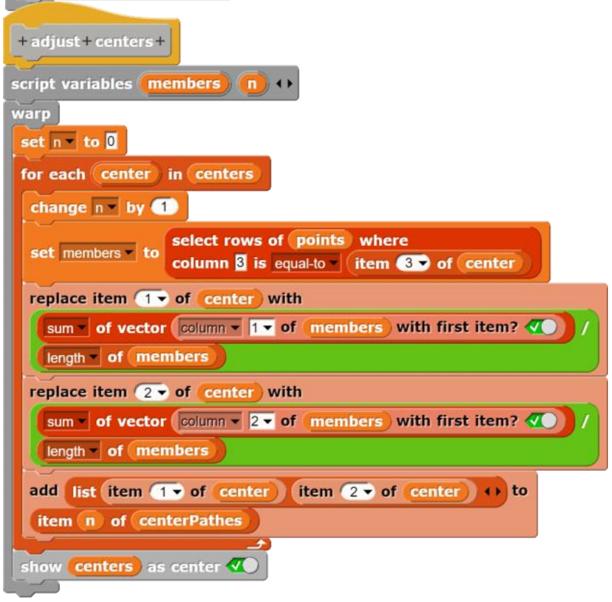
The k-means method now determines the nearest center for each point and colors the points in its color.



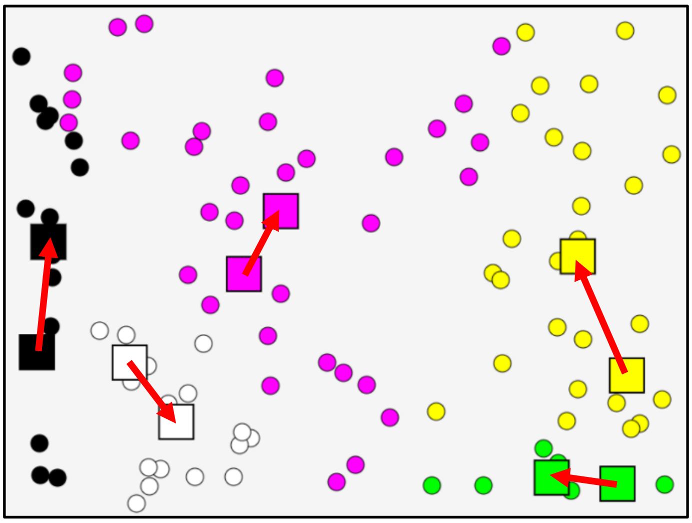
```
build clusters
script variables minDistance n nearestCenter distance
warp
  set anyChanges? to false
  for each point in points
    set minDistance to 1000000
    set n to 0
    set nearestCenter to 0
    for each center in centers
      change n by 1
      set distance to
        sqrt of ((item 1 of point − item 1 of center) × (item 1 of point − item 1 of center)
               + (item 2 of point − item 2 of center) × (item 2 of point − item 2 of center))
      if distance < minDistance
        set nearestCenter to n
        set minDistance to distance
    if nearestCenter > 0
      if item 3 of point ≠ item 3 of item nearestCenter of centers
        replace item 3 of point with
          item 3 of item nearestCenter of centers
        set anyChanges? to true
  configure theStage as an ImagePad width: 400
    height: 300 color: 245 245 245
  show points as center ☒
```



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.

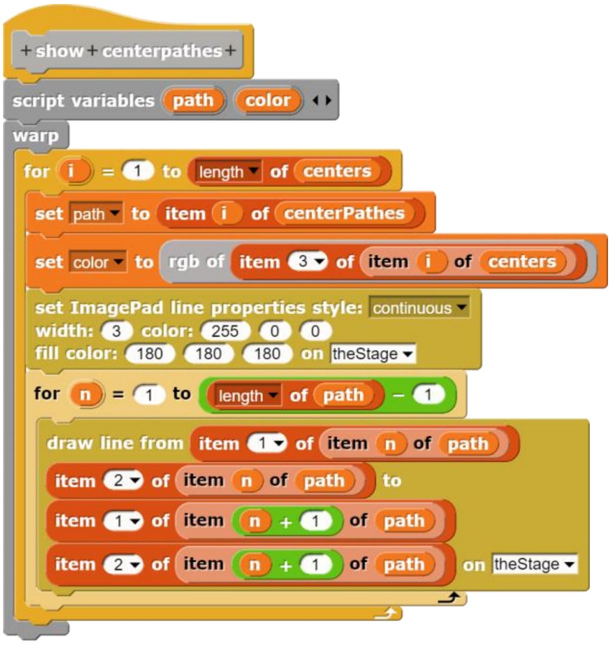


```
adjust centers
script variables members n
warp
  set n to 0
  for each center in centers
    change n by 1
    set members to select rows of points where
      column 3 is equal-to item 3 of center
    replace item 1 of center with
      (sum of vector column 1 of members with first item? ✓) /
      (length of members)
    replace item 2 of center with
      (sum of vector column 2 of members with first item? ✓) /
      (length of members)
    add list item 1 of center item 2 of center to
      item n of centerPathes
  show centers as center ✓
```

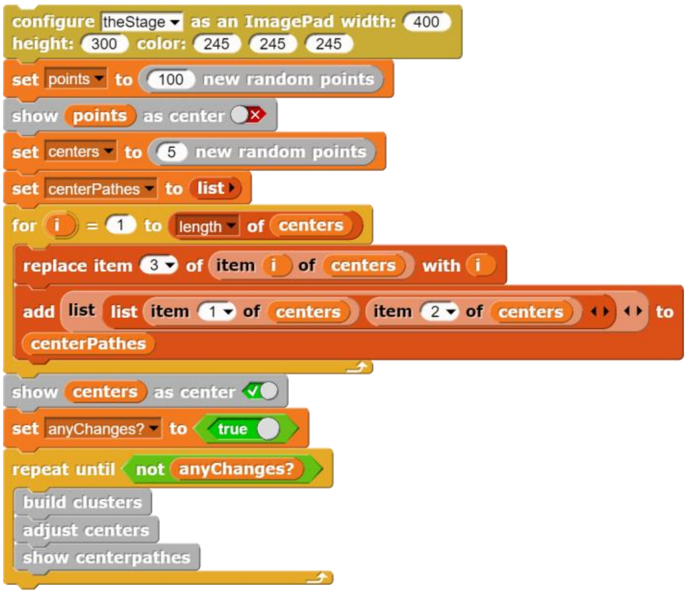


The procedure is continued until there are no more changes in the clusters. We plot the movements of the centers using the stored positions.

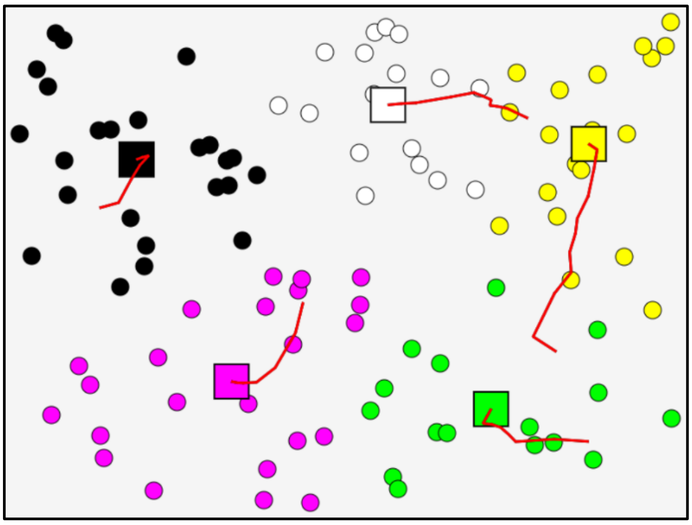
Weg get the result:



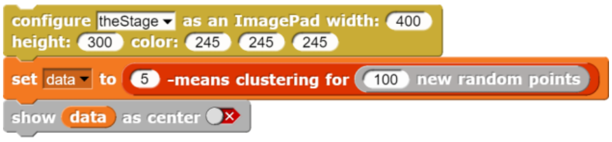
```
+show+centerpathes+
script variables path color ◂▸
warp
  for i = 1 to length of centers
    set path to item i of centerPathes
    set color to rgb of item 3 of item i of centers
    set ImagePad line properties style: continuous
    width: 3 color: 255 0 0
    fill color: 180 180 180 on theStage
    for n = 1 to length of path − 1
      draw line from item 1 of item n of path
      item 2 of item n of path to
      item 1 of item n + 1 of path
      item 2 of item n + 1 of path on theStage
```



```
configure theStage as an ImagePad width: 400
height: 300 color: 245 245 245
set points to 100 new random points
show points as center ✗
set centers to 5 new random points
set centerPathes to list ▸
for i = 1 to length of centers
  replace item 3 of item i of centers with i
  add list list item 1 of centers item 2 of centers ◂▸ ◂▸ to
  centerPathes
show centers as center ✓
set anyChanges? to true
repeat until not anyChanges?
  build clusters
  adjust centers
  show centerpathes
```



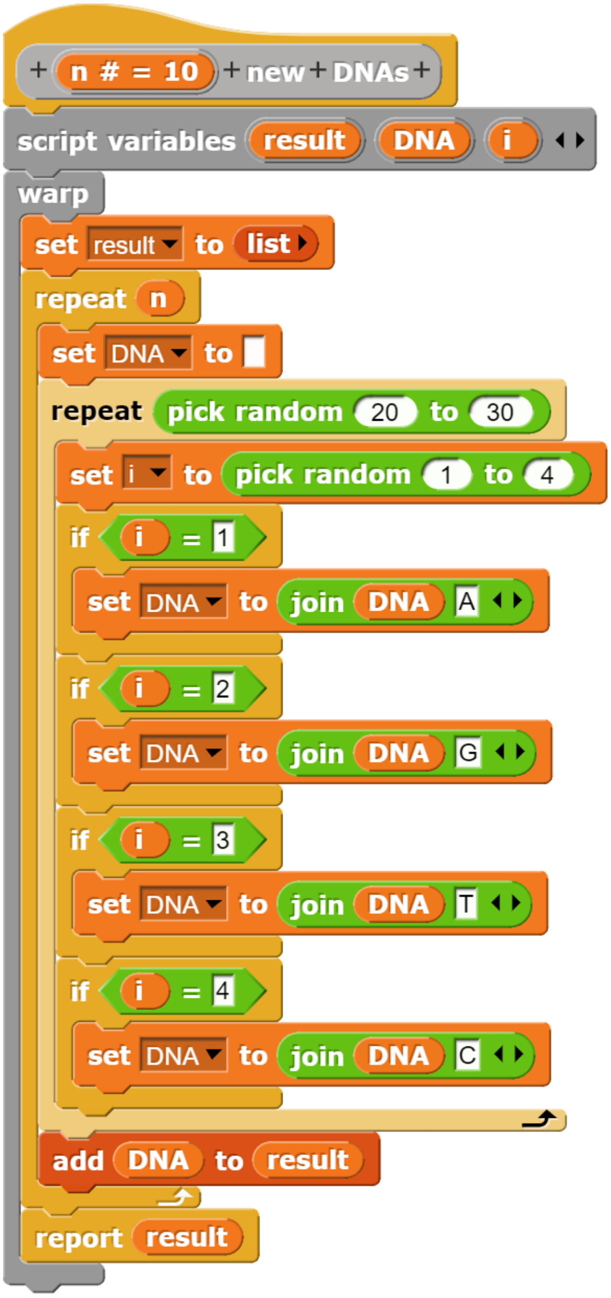
Using the existing blocks, we could of course have obtained the result a bit simpler - but just without representing the process:



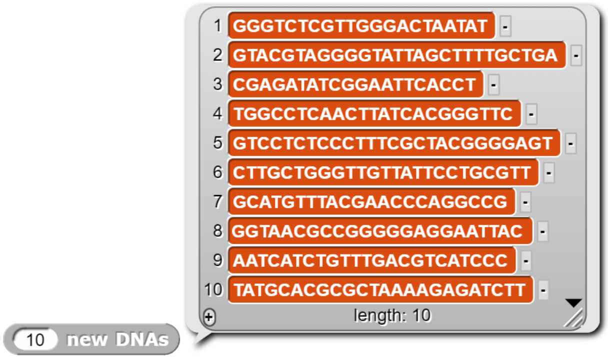
```
configure theStage as an ImagePad width: 400
height: 300 color: 245 245 245
set data to 5 -means clustering for 100 new random points
show data as center ✗
```

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.

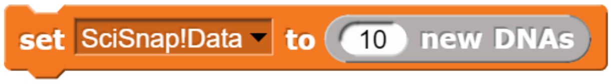


```
+ n # = 10 + new + DNAs +
script variables result DNA i
warp
    set result to list
    repeat n
        set DNA to
        repeat pick random 20 to 30
            set i to pick random 1 to 4
            if i = 1
                set DNA to join DNA A
            if i = 2
                set DNA to join DNA G
            if i = 3
                set DNA to join DNA T
            if i = 4
                set DNA to join DNA C
        add DNA to result
    report result
```



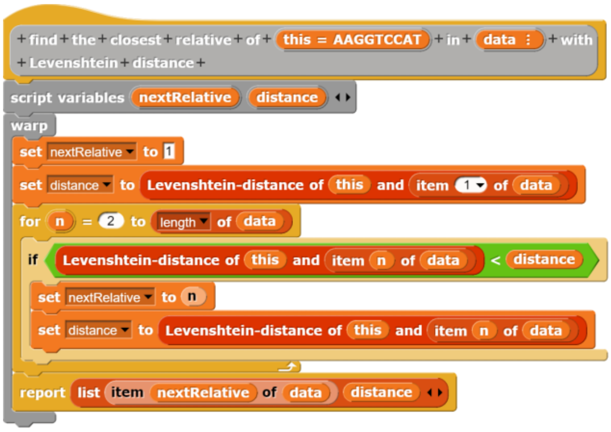
```
10 new DNAs
```

```
1  GGGTCTCGTTGGGACTAATAT
2  GTACGTAGGGGTATTAGCTTTTGCTGA
3  CGAGATATCGGAATTCACCT
4  TGGCCTCAACTTATCACGGGTTC
5  GTCCTCTCCCTTTCGCTACGGGGAGT
6  CTTGCTGGGTTGTTATTCCTGCGTT
7  GCATGTTTACGAACCCAGGCCG
8  GGTAACGCCGGGGGAGGAATTAC
9  AATCATCTGTTTGACGTCATCCC
10 TATGCACGCGCTAAAAGAGATCTT
length: 10
```

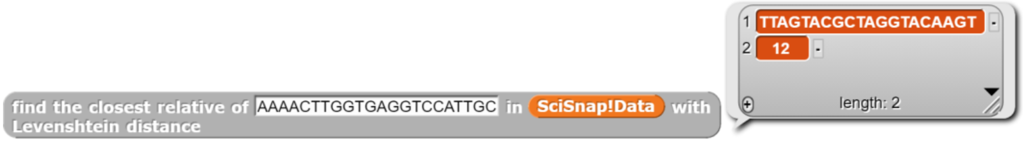


```
set SciSnap!Data to 10 new DNAs
```

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.



```
+ find + the + closest + relative + of + this = AAGGTCCAT + in + data ⋮ + with
+ Levenshtein + distance +
script variables nextRelative distance
warp
    set nextRelative to 1
    set distance to Levenshtein-distance of this and item 1 of data
    for n = 2 to length of data
        if Levenshtein-distance of this and item n of data < distance
            set nextRelative to n
            set distance to Levenshtein-distance of this and item n of data
    report list item nextRelative of data distance
```



```
find the closest relative of AAAACTTGGTGAGGTCCATTGC in SciSnap!Data with
Levenshtein distance
```

```
1 TTAGTACGCTAGGTACAAGT
2 12
length: 2
```

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!

References and sources

[ABELSON] Abelson, Sussman, Sussman: Structure and Interpretation of Computer Programs, MIT Press

[Census] https://archive.ics.uci.edu/ml/datasets/census+income

[DBV] Burger, W., Burge, M.-J-: Digitale Bildverarbeitung – Eine Einführung mit Java und ImageJ, Springer 2006

[FITS] de.wikipedia.org/wiki/Flexible_Image_Transport_System

[HOU] Hands-On Universe: handsonuniverse.org/

[HR] https://studylibde.com/doc/2985884/hertzsprung-russell--und-farb-helligkeits

[JSON] Popular Baby Names: https://catalog.data.gov/dataset/most-popular-baby-names-by-sex-and-mothers-ethnic-group-new-york-city-8c742

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

[SchulAstro] www.schul-astronomie.de

[SQL] Modrow, Eckart: Computer Science with Snap!, http://ddi-mod.uni-goettingen.de/ComputerScienceWithSnap.pdf

[UniGOE] Institut für Astrophysik, Universitaet Goettingen