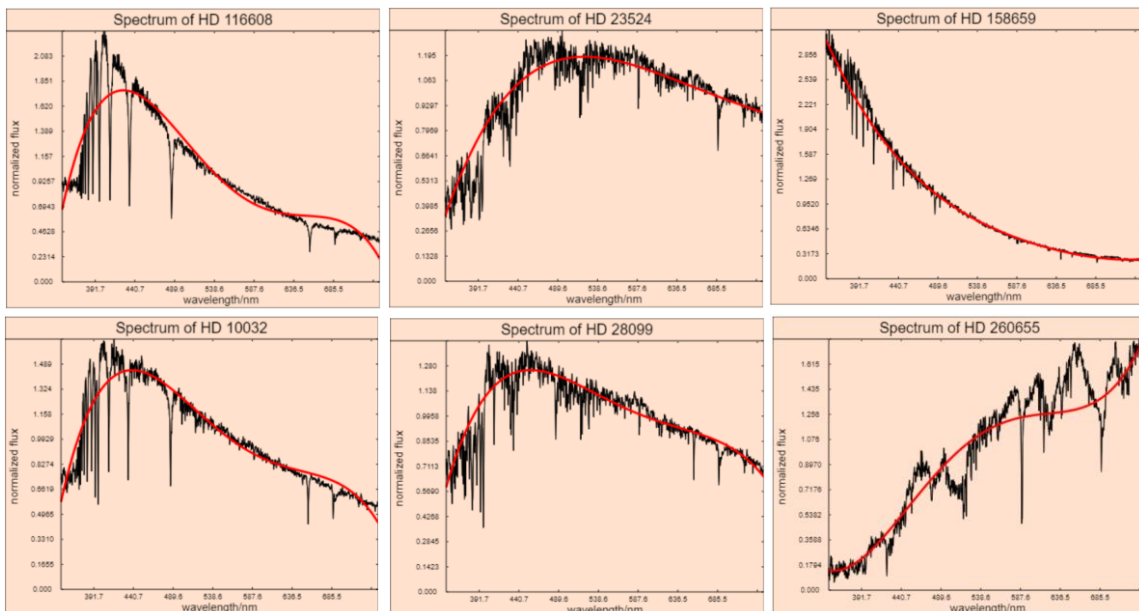


Eckart Modrow

# Machine Learning with the -DataSprite



© Eckart Modrow 2019  
emodrow@informatik.uni-goettingen.de



This work is licensed under a Creative Commons Attribution-Noncommercial-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 10 € to the PayPal account listed there. The scripts are developed with *Snap!* 5.0.8 *Build Your Own Blocks*.

The DataSpriteLibrary and this script can be loaded from

<http://emu-online.de/DataSpriteLibrary.zip> bzw.  
<http://emu-online.de/MachineLearningWithSnap.pdf>

---

Prof. Dr. Modrow, Eckart:  
Machine Learning with the *Snap!* - DataSprite  
© emu-online Scheden 2019  
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](mailto:emodrow@emu-online.de)  
Intended use: ML-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

This script describes the *DataSpriteLibrary* with *Snap!* blocks, which is intended for (relatively) fast processing of large amounts of data. "Large" data volumes are almost never used in schools and initial university education - because they were hardly freely available some time ago, and money is scarce in education. In the meantime, however, there are large amounts of data in abundance, be it as a data collection on the Internet or as image files, because they are also "large". Education thus has the chance to deal with relevant data and thus find numerous points of contact with the field of "computer science and society". In the long run they are more important than any programming tricks in terms of general education.

Especially for beginners it is important to "see" what they are doing with their programming attempts. *Snap!*'s fantastic visualization capabilities are complemented by the *DataSpriteLibrary*, which includes library functions for graphics and images that, like the *Snap!* tables, quickly display the results of operations. Speed is important in this area because it supports experimental work in trial and error style. If you must wait too long, you won't try that much. The *DataSpriteLibrary* supports this approach by implementing most time-critical functions in JavaScript. Besides, these blocks also show how text-based programming can be senseful integrated into a graphical development environment.

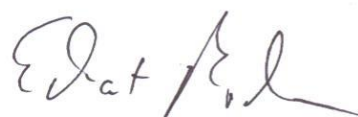
The *DataSpriteLibrary* contains blocks from the area of data visualization and table handling, which is supported by the introduction of the data type *table*. In addition, functions of linear algebra with the data types *vector* and *matrix*, the solution of linear systems of equations and interpolation by polynomials are available. Image operations can be performed quickly using kernels as well as vector and matrix blocks. The examples show how this can be done. But they always show only one way - invent others and better ones for yourself!

This book is a translation from German. Unfortunately, I do not speak English well, so it will be bumpy. I apologize for that. Be strong and hold it! Many thanks for the wonderful help of the *DeepL*<sup>1</sup> translation program. I would probably never have finished without these.

I would like to thank Jens Mönig and Rick Hessman very much for their support and the numerous discussions.

Otherwise, I hope you enjoy working with *Snap!* and the *DataSpriteLibrary*!

Goettingen, 6 August 2019



---

<sup>1</sup> <https://www.deepl.com/translator>

# Content

Preface .....	3
Content .....	4
1 Artificial Intelligence and School .....	5
2 Machine Learning .....	7
3 The Structure of the DataSprite .....	9
4 Working with the DataSpriteLibrary .....	10
4.1 Creating DataSprites .....	10
4.2 Importing data .....	10
4.3 Exporting data .....	12
4.4 Importing data with the mouse .....	13
4.5 More blocks in the Sensing palette .....	14
4.6 The properties of the DataSprite .....	14
4.7 Graphics on the DataSprite .....	15
4.8 Working with DataSprite tables .....	17
4.9 Working with DataSprite operators .....	19
5 Applications of the DataSprite .....	22
5.1 Under- and Overfitting .....	22
5.2 New York Citibike Tripdata .....	25
5.3 Star spectra .....	29
5.4 Period of a Cepheid .....	32
5.5 Search for a Supernova .....	35
5.6 Classification of stars according to the kNN method .....	38
6 Hints .....	40
References and sources .....	41

# 1 Artificial Intelligence and School

The term "artificial intelligence" is currently and for the foreseeable future more than current. In Germany, the Year of Science 2019 has been declared the "Year of Artificial Intelligence". In the field of "digitalisation", the term is shaping discussions in the media, business and politics. Informatic didactic contributions are also increasingly being made on the subject.

In school informatics the topic is not really new. For three to four decades now, there have been examples of neural networks (NNs) suitable for use in schools, for example, which are developed and trained by the pupils themselves [Baumann] [Modrow1]. Such networks are clear and easy to understand, encourage students to work independently and then to discuss philosophical implications based on their professional experience [Modrow2]. Above all, however, they are small. This is exactly the difference to the current NNs: they are big. According to Ian Goodfellow [Deep Learning], one of the leading developers in this field, basically nothing has changed compared to the old small networks. The structure and methods have (almost) remained the same, but of course they have been improved. What has changed is the performance of the computers on which the NNs run and the amount of data available to train them. This, however, leaves older findings valid, such as Marvin Minsky's [Minsky] 1967 findings on the equivalence of NNs and finite automata. The result is no wonder, because the model of finite automata has its roots in the first NNs.

The suggestions for treating large NNs in class often consist of training finished NNs using finished training data. Students then watch the net learn, slowly improving its results. Actually, you don't need a real NN for this experience, a video was enough. You can't see that the net is big and you can't see why this size is important from watching it. All you can see is that the results are improving. You don't learn anything from this experience alone from NNs. A discussion of the effects of NNs then is based on the information that they exist and that they can learn. Further technical basics are missing, so that this discussion could take place just as well in other subjects.

Let us compare the situation with an example from physics. The relatively new image of a black hole [SZ] shows that there are black holes and that they "swallow" matter. However, this information alone does not integrate the topic into the physics lesson, because a technical treatment of black holes is largely beyond the possibilities of the school. But within a subject area "gravitation", which contains numerous activities, historical and social references, typical problems of school physics, etc., the picture links school physics with "science after school", shows ways to a more profound occupation with it and, for example, encourages reflection on whether the learners see a personal perspective in this area - or not.

What do we learn from this?

The pure introduction of new technologies has no place in school - there are other channels for shows. The pure information that such technologies exist is also not enough to assign the topic to a specific subject. On the contrary, if you limit yourself to that, then it would be better to locate subjects in which, for example, the social or philosophical effects are discussed, and the topic is thus networked with other aspects. Only the didactic reduction

of a question to a complexity level, on which the learners can work as independently and imaginatively as possible, makes the topic pedagogically fruitful.

*In the field of artificial intelligence, it is not the passive observation of the learning of networks in schools that is important, but the active promotion of the understanding of human learners for the fundamentals and implications of this process.*

One more note: If the school concentrates on conveying facts and data and practicing the application of calculae, then in my opinion this presupposes that the learners are not able to discover and understand connections and backgrounds themselves. The procedure therefore promotes immaturity. And even worse: it keeps learners in immaturity because they learn one thing for sure: that they are not expected to think for themselves.

## 2 Machine Learning

The term "machine learning" is often used as a synonym for "artificial intelligence" or "neural networks". However, this limitation is not true. For example, the definition found on the SAP page [SAP] is more precise:

*Machine learning technology teaches computers to perform tasks by learning from data instead of being programmed for the tasks.*

"Learning from data" can be understood as adapting the parameters of a function. A data set (image, table, character string, ...) is presented as input vector  $E$  to a machine. It calculates an output value  $k$  from this, which assigns the input to a category ("It is a cat", "Feature present" (or not), "The word 'car'", ...).

$$f(E) = k$$

This assignment can take place in very different ways. For example, you can adjust the parameters of a polynomial, search for similar input values ("k-next neighbours"), work with decision trees, use Bayesian filters, ... - or even train an NN. All these methods have in common that the "machine" contains a set of parameters that can be changed. The machine "learns from data" by repeatedly reading in a data set, calculating the output value from this using the current parameter set, and then comparing this output with the "desired" output value using some method. If there is a deviation, it changes the parameters so that the output at least approaches the "desired" value. "Desired" values may be known in advance ("The image is a cat image"), may come from outside e.g. from a "trainer" ("supervised learning") or may be generated by the machine itself ("unsupervised learning"), e.g. by extracting features from many training data ("clustering"). In all cases, the machine does not "learn" anything, but adapts parameters according to a given procedure.

This approach, too, has long been widespread in schools. "Learning Nim-games" etc. can already be found in the first computer science textbooks. What is new again is the scope of the required training data. A large NN can have billions of parameters that need to be trained - and this requires "a lot of" training data. Another new feature is that these data are available on the net. So, if applications available "for free" are paid "with data", then we now also know how and why this happens.

If you look at common textbooks on machine learning [Grus] [Albon], you won't find much about NNs, but a lot about data handling. These have to be normalized, for example, in order to make the many input data, which can come from very different sources, compatible. For example, if we photograph many dogs with an older digital camera and many cats with a newer one, then an NN would very likely learn from these images that dog images are smaller than cat images.

The preparation of data now is a very manual activity. It can be done step by step, tested and then automated with simple algorithms. Testing is greatly facilitated if the structure of the data is easy to visualize, e.g. in tables or as a graph. And algorithms are simple if they have a clear structure, e.g. if, after some preparation steps, they consist of a loop in which some alternatives with the corresponding instructions are enumerated. The power of the developed scripts does not depend so much on the algorithmic structure as on the power of the available commands. Or vice versa: if you have enough powerful commands, you

---

can do a lot with simple programs. The parameters then can be adjusted in one of the usual ways. If the appropriate tools are available, the preparation of data is a very suitable topic for schools. The *DataSprite* is intended as such a tool.

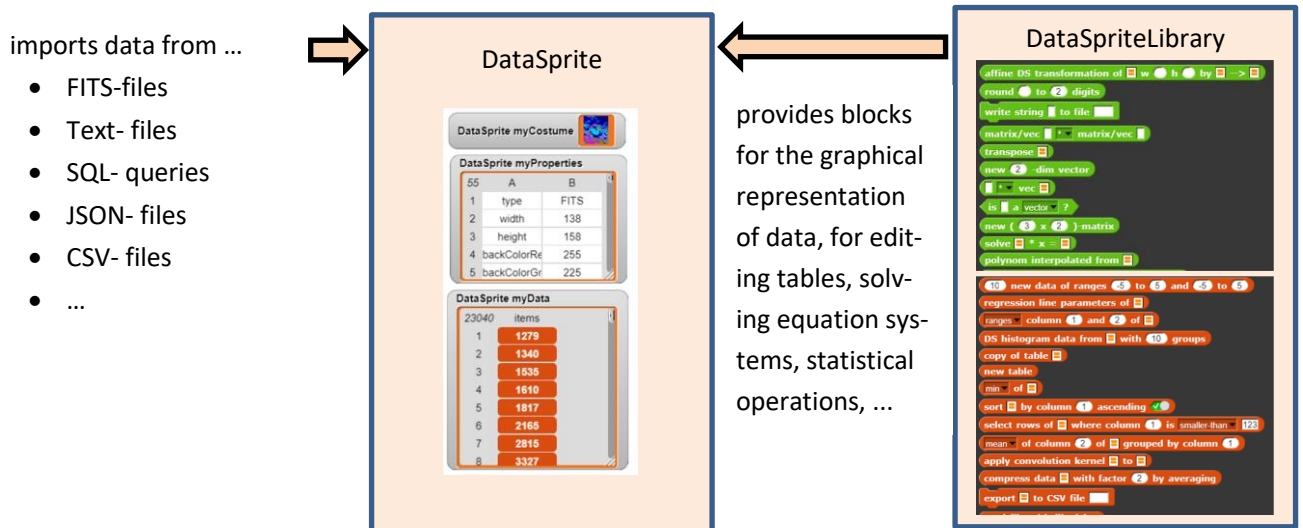


### 3 The Structure of the DataSprite

The structure of the *DataSprite* is based on the idea of documented data sets consisting of two parts: the *metadata*, which describes the structure and context of the data (e.g. number format, image dimensions, recording device, recording date, ...) and the associated pure *data* segments. Metadata usually consists of dictionaries - names with assigned values (e.g. "Recording date: 24.12.2018"). Examples for this structure are FITS files [FITS], which are standard in astrophysics but are also used in the Vatican Library, or JPEG images from mobile phones. Also, here there are meta data (image size, compression degree, date of acquisition, often also GPS coordinates). Without these an image generation would not be possible. It is important that the image generation does not change the original data.

We adapt this structure by giving a *DataSprite* three local variables containing the data (*myData*), the data description (*myProperties*) and the current costume (*myCostume*). These variables can be filled by importing data from different sources (SQL queries, text file, CVS file, JSON file, FITS file, direct assignment, ...), whereby the properties *myProperties* have to be adapted to the respective data. 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. Because tables can be displayed very nicely in *Snap!*, this display format is not additionally implemented. Therefore, the data type *table* is implemented with many of the operations commonly used in *data science* (table operations, correlation calculation, affine transformations, solving linear systems of equations, ...), which can handle larger amounts of data sufficiently quickly.

The overall structure is as follows:



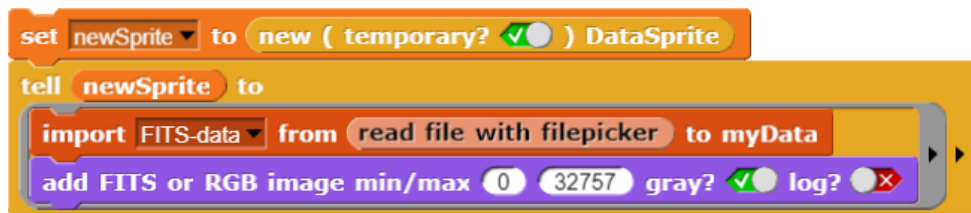
## 4 Working with the DataSpriteLibrary

### 4.1 Creating DataSprites

When we work with data and graphics, it makes sense to use multiple *DataSprites* at the same time. If we already have one, we can create more by cloning or copying. Of course, we also copy the remains of the previous work (variables, new blocks, ...), and perhaps the three local variables of the sprite are missing, are named differently, ... There are many possible errors. To avoid that, there is a new reporter block *new (temporary?) DataSprite* in the Commands palette. It creates a new sprite, either permanent (visible in the sprite coral) or temporary (it is automatically deleted when the red button is pressed or when *Snap!* is closed). The block is a reporter because the new sprite has to be accessed from outside very often. You should save a reference to it in a variable.

**new ( temporary? ☒ ) DataSprite**

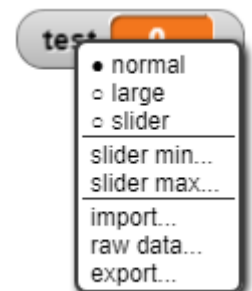
Example: A FITS file (source: [HOU]) is read in and displayed on a new *DataSprite*.



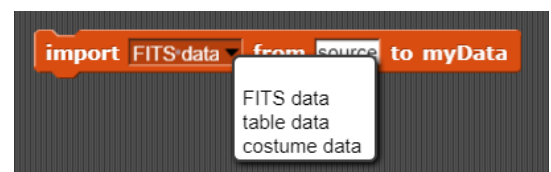
### 4.2 Importing data

*Snap!* can import a variety of data formats directly. This can be done by dropping files onto the *Snap!* window or by right-clicking on a variable watcher to import them. Both works well with text, CSV and JSON files. Other text file formats like FITS can also be imported in this way, asking if they are serious. Exporting works in the same way. If you want to do the same by programs, use the reporter block *read file with filepicker*. A file manager window appears in which you select the file as usual. Then the data will be imported.

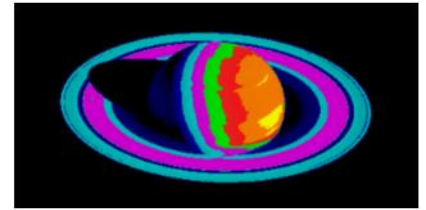
The main task afterwards is to assign this data to the *myData* variables and set the corresponding properties in *myProperties*. This is done by the following block, which imports data from outside into the *myData* area. This can be FITS data, table data or the data of the current costume. This is stored as a table of RGB values.



**read file with filepicker**



Example: An image (source: [NASA]) is saved and re-displayed with false colors.



Example: Almost 600 000 data records from a CSV file are read in about 10 seconds. The properties are set.



	A	B
34	A	B
1	type	table
2	width	0
3	height	0
4	backColorRe	255
5	backColorGr	225

	A	B
577704	A	B
1	tripduration	starttime
2	695	2013-06-01 (2013-06-01 00:00:00)
3	693	2013-06-01 (2013-06-01 00:00:00)
4	2059	2013-06-01 (2013-06-01 00:00:00)
5	123	2013-06-01 (2013-06-01 00:00:00)

Example: SQL-Import

If we have access to a SQL server, then we can also read data from there. In our case we import a *SQL server sprite* and the library with the SQL blocks [SQL] into a *Data-Sprite*. Then we ask the SQL server to establish a connection to a school database.



From this we would like to query the names, the gender and the middle grade of English. This will give us a list of strings containing these data - separated by commas. So, we have to convert the list entries into sublists and enter them into a table. After that we can import these in the sprite.



answer

```

1 Aehrlich,Hanna,w,7.5000
2 Antoln,Max,m,4.5000
3 Bahn,Johann,m,6.2500
4 Batton,Alina,w,11.2500
5 Benner,Janina,w,10.7500
6 Berg,Leni,w,6.2500
7 Beusberg,Christina,w,7.7500
8 Boemmel,Hugo,m,4.2500
9 Brummel,Otto,m,9.2500

```

length: 83

table

83	A	B	C	D
1	Aehrlich	Hanna	w	7.5000
2	Antoln	Max	m	4.5000
3	Bahn	Johann	m	6.2500
4	Batton	Alina	w	11.2500
5	Benner	Janina	w	10.7500
6	Berg	Leni	w	6.2500
7	Beusberg	Christina	w	7.7500
8	Boemmel	Hugo	m	4.2500
9	Brummel	Otto	m	9.2500
10	Brummer	Steffi	w	6.0000

DataSprite myData

83	A	B	C	D
1	Aehrlich	Hanna	w	7.5000
2	Antoln	Max	m	4.5000
3	Bahn	Johann	m	6.2500
4	Batton	Alina	w	11.2500
5	Benner	Janina	w	10.7500
6	Berg	Leni	w	6.2500
7	Beusberg	Christina	w	7.7500
8	Boemmel	Hugo	m	4.2500

DataSprite myProperties

34	A	B
28	scalesPrecis	3
29	scalesTexthe	12
30	scalesNumb	10
31	minValue	not set
32	maxValue	not set
33	columns	4
34	rows	83

Example: JSON-Import

The easiest way is to simply "drop" a JSON file into the *Snap!* window. But it can also be automated. First of all, we look for interesting JSON data and of course choose the statistics of baby names in New York City - what else? The suitable block is again *import <table data> from <read file with filepicker> to myData*. The result is a list with two columns and two rows, the metadata and the actual data. Because we are interested in them, we replace the original data with the element (2|2) of the table. From the many columns we copy the three interesting ones into a new table, add column headings and import the result back into *myData*.

The image shows a Scratch script on the left and a DataSprite window on the right. The script performs the following steps:

- Import table data from read file with filepicker to myData
- Set myData to item 2 of item 2 of myData
- Set table to new table
- Add column 10 of myData to table
- Add column 12 of myData to table
- Add column 13 of myData to table
- Add column headers list gender name number to table
- Import table data from table to myData

The DataSprite window titled "DataSprite myData" shows a table with 15 rows and 4 columns (A, B, C). The data is as follows:

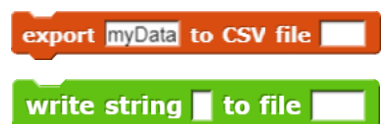
	A	B	C
1	gender	name	number
2	FEMALE	Olivia	172
3	FEMALE	Chloe	112
4	FEMALE	Sophia	104
5	FEMALE	Emily	99
6	FEMALE	Emma	99
7	FEMALE	Mia	79
8	FEMALE	Charlotte	59
9	FEMALE	Sarah	57
10	FEMALE	Isabella	56
11	FEMALE	Hannah	56
12	FEMALE	Grace	54
13	FEMALE	Angela	54
14	FEMALE	Ava	53
15	FEMALE	Joanna	49

The result: 19419 baby names.

Who would have thought!

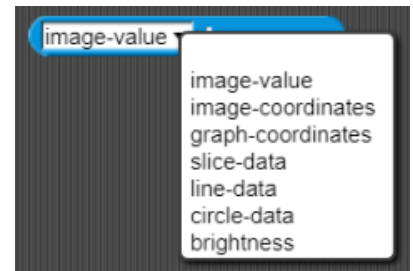
### 4.3 Exporting data

The export again can be done directly from a variable watcher. For scripts there are two new blocks *export <table> to CSV file <filename>* and *write string <string> to file <filename>*. As usual in *Snap!*, the results will be saved in the download folder of the browser. The two blocks allow you to automate data exchange with spreadsheet programs or text files, for example to save data processing results.



## 4.4 Importing data with the mouse

In many cases it is advantageous to read data using the mouse. The `<...> by mouse` block is available in the Sensing palette for this purpose. It can be used to determine image values, image coordinates, coordinates in the coordinate system used for graphs and/or data points, the data on a slice through the image, start and end points of a line, center and radius of a circle and the summed screen values together with their number in a circle. As an example, a section through a moon image is shown.



Example: Slice through an image (Source: [HOU])

import **FITS-data** from **read file with filepicker** to **myData**

add **FITS or RGB image min/max** **0** **32757** **gray?** ☒ **log?** ☐

set **data** to **slice-data by mouse**

	A	B
149		
1	0	2047
2	1	1877
3	2	2303
4	3	1791
5	4	2901
6	5	3182

Example: Measuring the sum of the image values within the radius (Source: [HOU])

import **FITS-data** from **read file with filepicker** to **myData**

add **FITS or RGB image min/max** **min** of **myData** **max** of **myData**

**gray?** ☐ **log?** ☒

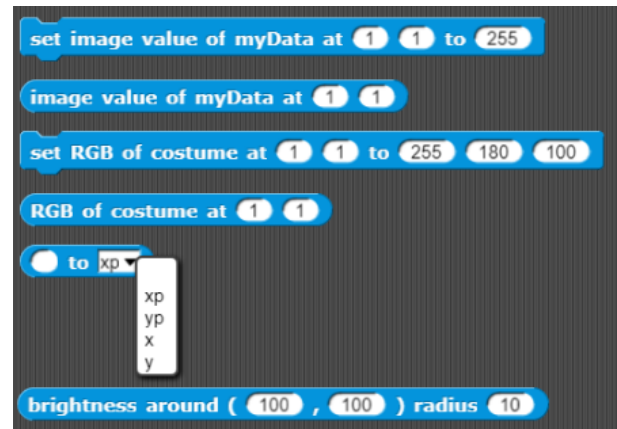
set **data** to **brightness by mouse**

1	262172
2	489

length: 2

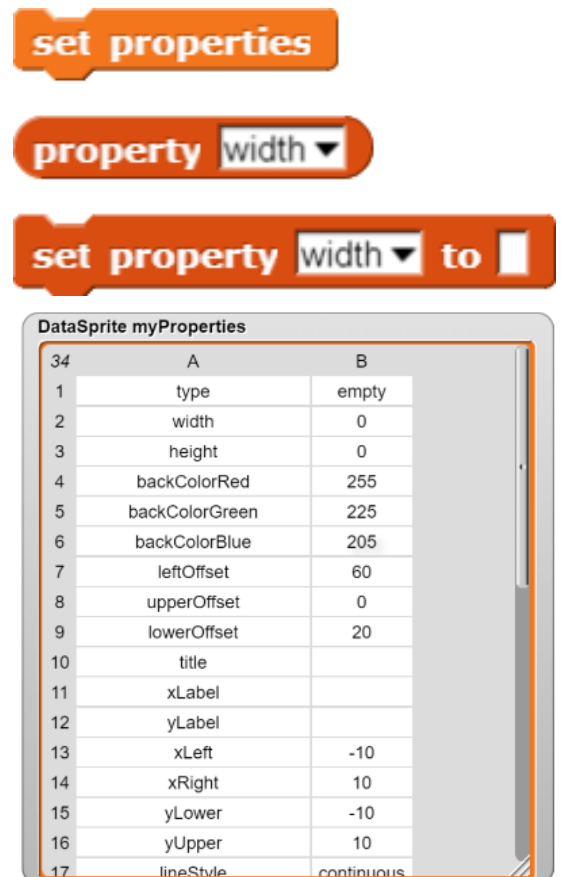
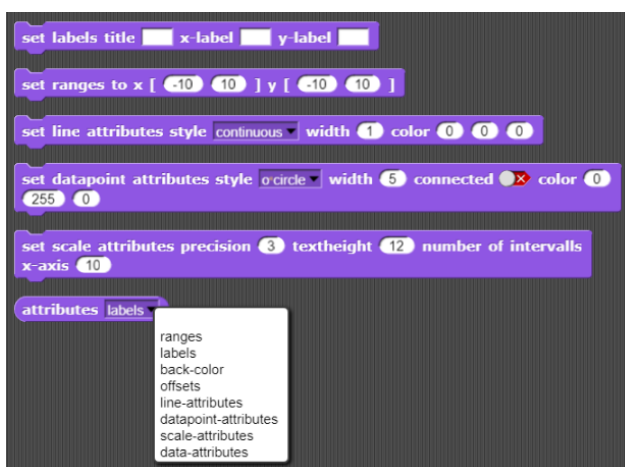
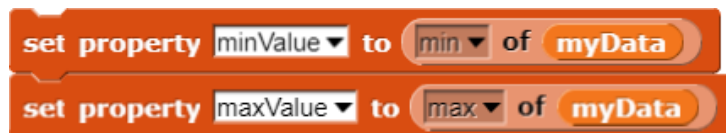
## 4.5 More blocks in the Sensing palette

The following blocks can be used to read or set the image value at a position or the corresponding pixel value of the costume. The penultimate block is used for conversions between the coordinate systems. The brightness in the periphery of a point can be determined with the last block.



## 4.6 The properties of the DataSprite

The Looks palette contains several blocks for displaying data on the *DataSprite*. They use the *myProperties* settings for value ranges, colors, sizes and line types, axis labels, and so on. These can be set to initial values with the *set properties* block and displayed directly as a *Snap!* table. If you are satisfied with them (as in the previous examples), you can draw graphics directly onto the *DataSprite*. Otherwise you must change the settings. This can be done directly with the two blocks for reading property values *property <property>* and for writing *set property <property> to <value>*. A typical use case would be to enter the range of the image values - if this has not already been done automatically.



This can be done somewhat more comfortably with the blocks that combine groups of properties - for setting or reading. The second simplifies the call of JavaScript functions because the number of parameters is somewhat limited.

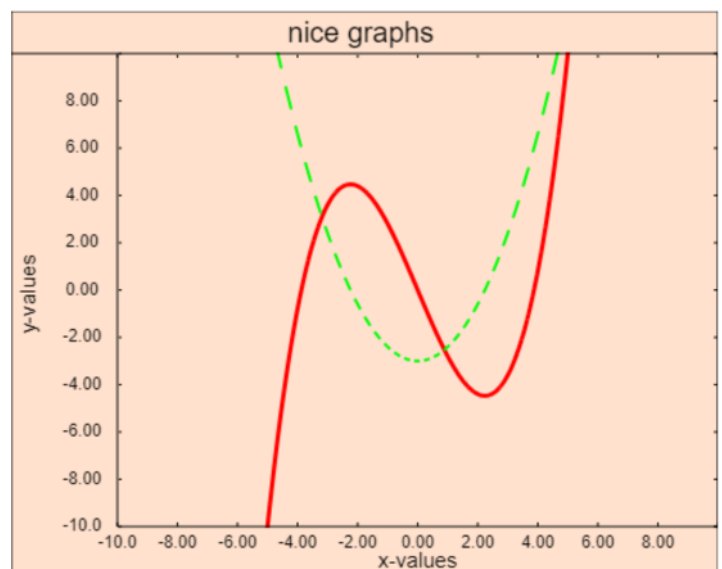
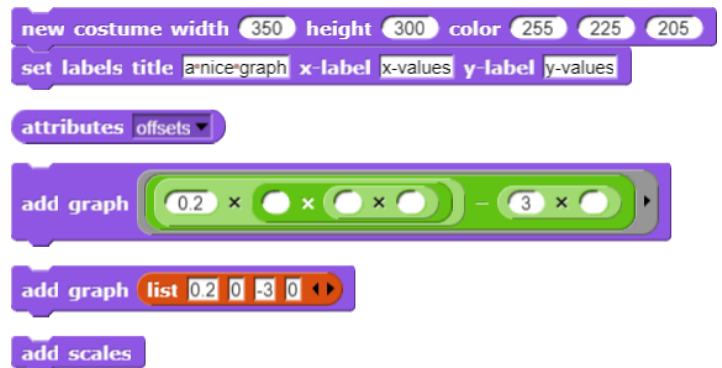


## 4.7 Graphics on the DataSprite

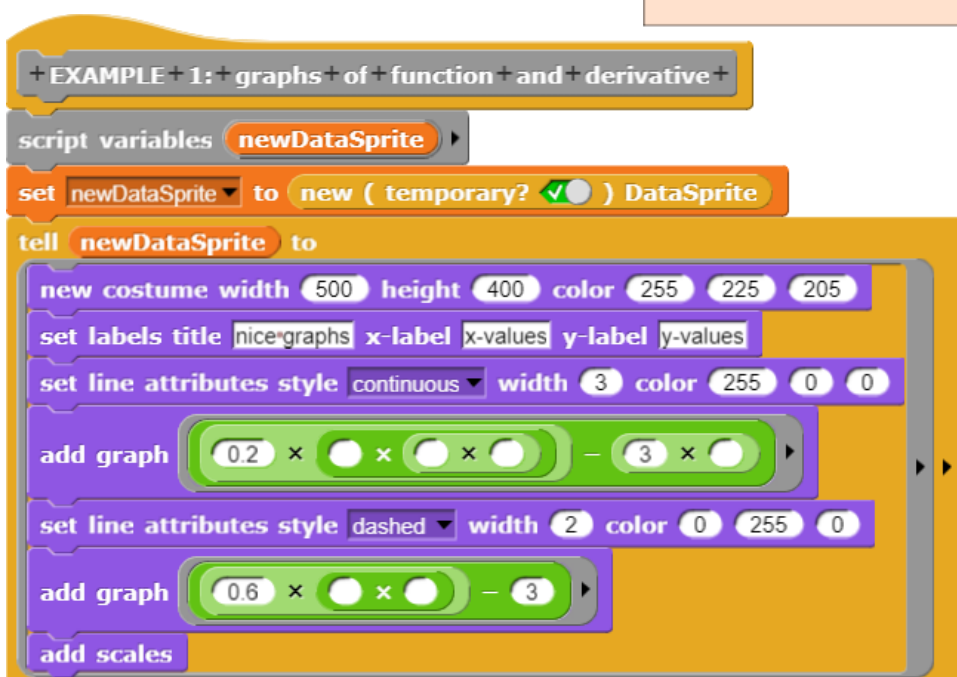
Graphics can be superimposed on *DataSprites*. As an example, a data plot can be used for which an approximation function is sought. You can experiment with it until you are satisfied with the result. If the image is "full", you simply draw a new one.

First of all you need a *DataSprite*. Size and background color can be set to make everyone happy. While we're at it, we also specify a chart title and axis labels. Since these require space on the diagram, the *offsets* are set so that the pure diagram area is slightly reduced. If required, we can also display them with *attributes <offsets>*.

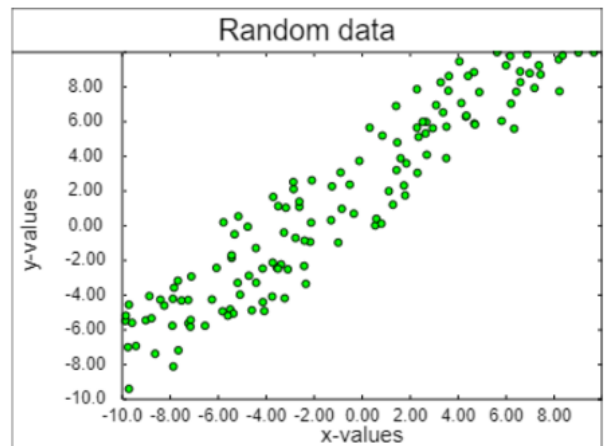
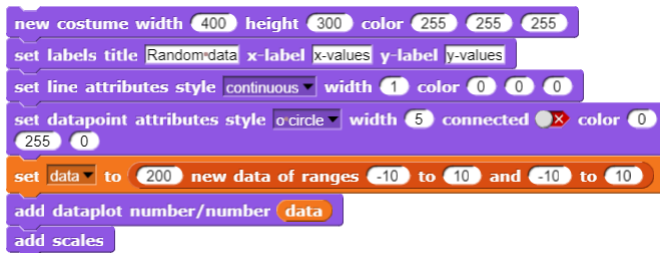
Next we use the block *add graph <term>* to draw a function graph. As term we can pass either a *Snap!* operator (*ringified* so that it is not executed before the call!) or the coefficient list of a polynomial. Further graphs - here: the derivative - can follow. The drawing of the axes and the labels is done by the block *add scales*.



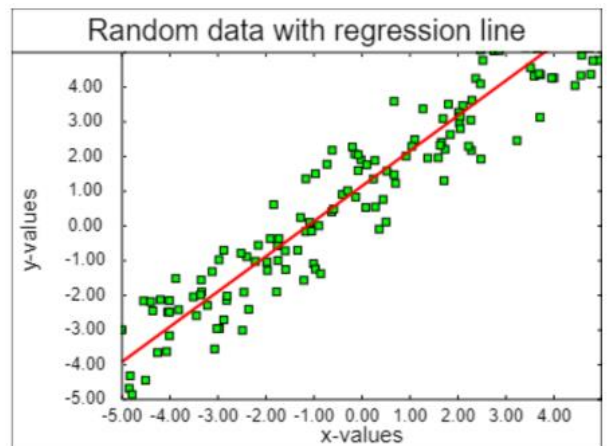
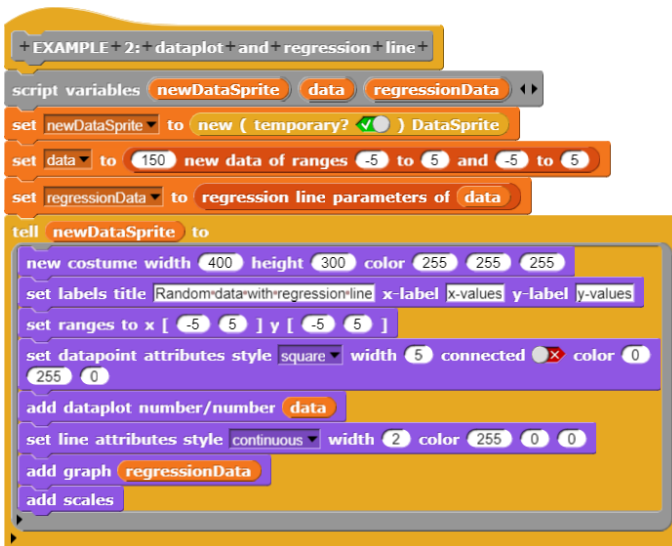
Of course, we can also create a new *DataSprite* instead and tell it what to draw.



If we want to display the contents of a data table graphically, this can be done with the block *add dataplot number/number*. Scales and labels are supplemented again with the block *add scales*. The type of display can be adjusted very precisely with the block *set datapoint attributes* ....

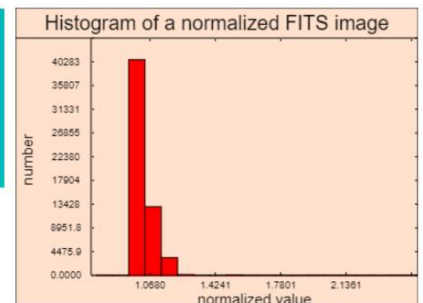
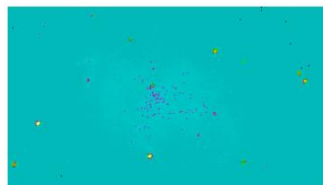
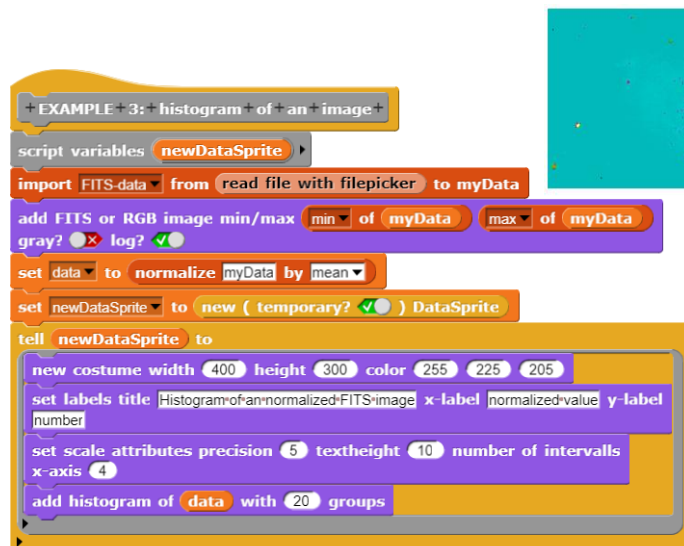


Example: A data set is represented with the corresponding regression line in a *DataSprite*.



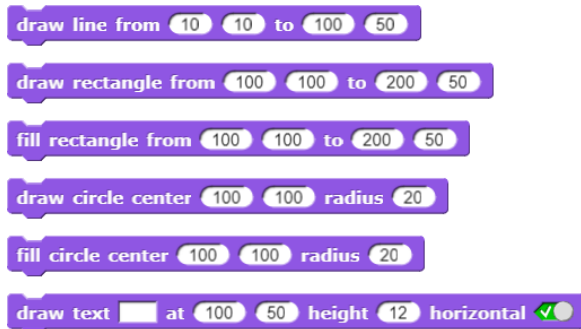
Histograms can be generated and displayed directly from data sources.

Example: A FITS image is loaded, the normalized distribution of the image values is displayed as a histogram on a new *DataSprite*.





In addition, the Looks palette contains six blocks for drawing basic structures on the sprite surface (not on stage).



## 4.8 Working with DataSprite tables

Because *Snap!* can display tables (lists of lists) so well (*right click on the variable watcher, open in dialog, adjust column width by dragging with pressed mouse button in the column header*), imported, modified, differently generated, ... tables can be displayed immediately. The results of working with tables can be checked immediately - a very important feature for interactive working with data.

The *DataSprite* therefore contains a data type *table* that corresponds to such a two-dimensional table.

A new empty table is simply an empty list. You can add rows, columns, and column headings to it, and you can delete them. If necessary, you can create a two-column table directly from random numbers (see above). In many cases *myData* is preset as the table to be edited. You can overwrite this entry by inserting a table variable.

If you need a real copy, i.e. data without references to other data, you can do this with *copy of table <tablename>*. You can also read out individual rows or columns of a table.

With single table columns you can also do some things: they can be normalized by dividing all entries by the mean value, you can sort them ascending or descending and calculate minimum, maximum, number, sum, mean, median, variance and standard deviation of the table values quickly.

If you need random pairs scattering around a given function graph, you can use the block *<n> random points near <operator> between <xmin> and <xmax>*.



set data to 10 new data of ranges -5 to 5 and -5 to 5

	A	B
10		
1	-3.03	-2.264998411809634
2	3.61	3.137411605194739
3	-1.7	-0.9981749908373895
4	4.36	5.1608035274100845
5	-1.09	-0.16162265178507296
6	0.86	0.6503976912335762
7	-4.7	-3.701562852454529
8	2.02	3.2379609770162228
9	3.13	4.7912180102604065
10	-3.96	-4.231492943213595

OK

set data to new table

add row myData

row  
column  
column headers

delete row 1 of myData

copy of table

row 1 of myData

row  
column

normalize myData by mean

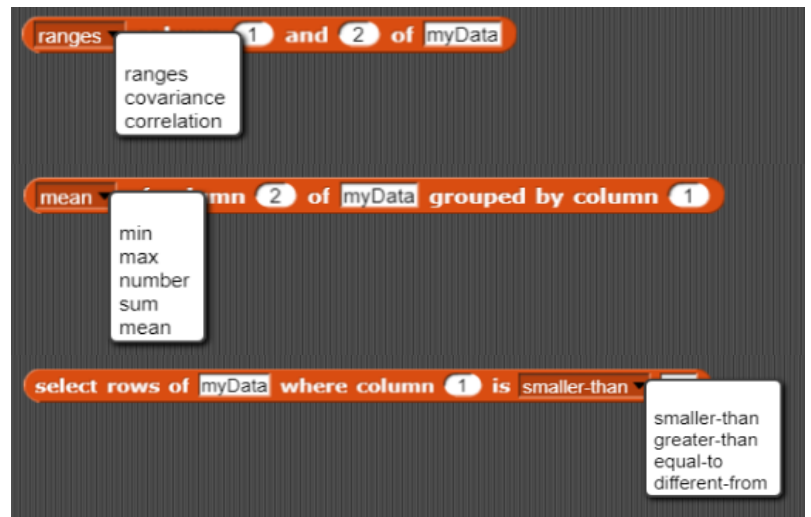
mean  
max  
number  
sum  
median

sort myData by column 1 ascending

min of

min  
max  
number  
sum  
mean  
median  
variance  
standard-deviation

The references between several table columns of course are also of interest. The value ranges, the covariance and the correlation can be calculated for two table columns. Some standard values can be grouped according to the entries in another column and table rows with predefined properties can be selected.



Example: A table with 100 rows and 10 columns is filled with random numbers. Afterwards, it should be determined between which columns the highest correlation exists.

**data**

	D	E	F	G	H
1	89	52	27	52	33
2	36	70	88	12	25
3	10	83	7	76	8
4	16	41	35	33	92
5	69	31	94	31	62
6	79	20	35	34	38
7	34	54	89	93	64
8	52	68	5	18	4
9	64	11	55	35	56
10	87	19	63	11	34
11	59	88	4	69	93
12	23	39	9	92	20
13	24	58	45	10	33
14	4	26	12	7	78
15	63	32	30	93	94
16	49	72	57	52	65
17	12	59	84	62	69
18	69	58	49	30	82
19	42	60	98	54	68
20	6	66	16	51	95
21	1	44	60	14	50
22	90	...	...	...	36

**correlations**

	A	B	C
1	6	10	0.2161585647
2	1	2	0.1721997030
3	5	10	0.1681766401
4	3	6	0.1570865552
5	2	10	0.1410832429
6	2	8	0.1392892957
7	1	10	0.1359997948
8	7	9	0.1313332775
9	6	8	0.1279894316
10	6	9	0.0989424889
11	1	3	0.0954672804
12	2	4	0.0848204148
13	8	10	0.0712208978
14	4	9	0.0691325904
15	4	7	0.0674427719
16	2	6	0.0502528787
17	2	7	0.0496690263

If necessary, convolution can be applied to image data using kernels, data rows can be compressed, and the nearest neighbors of a new data tuple can be determined (kNN).

apply convolution kernel  to myData

compress data myData with factor 2 by averaging

5 next neighbors of  in myData

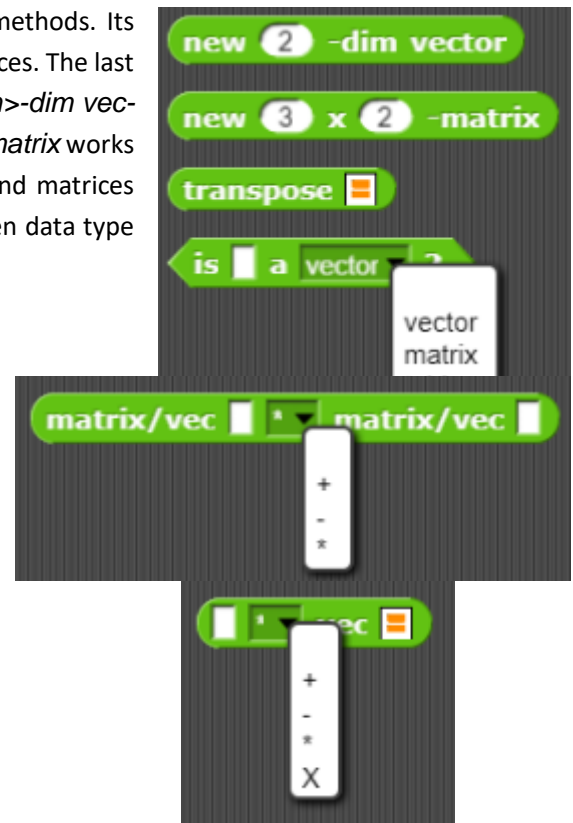
## 4.9 Working with DataSprite operators

Machine learning often requires the use of linear algebra methods. Its basic elements include scalars ("numbers"), vectors and matrices. The last two can be generated quickly with random contents: *new <n>-dim vector* returns a vector of the given length and *new <n>x<m>-matrix* works accordingly for matrices. With *transpose <data>* vectors and matrices can be transposed, *is <data> a ...* checks, whether the given data type has the correct structure.

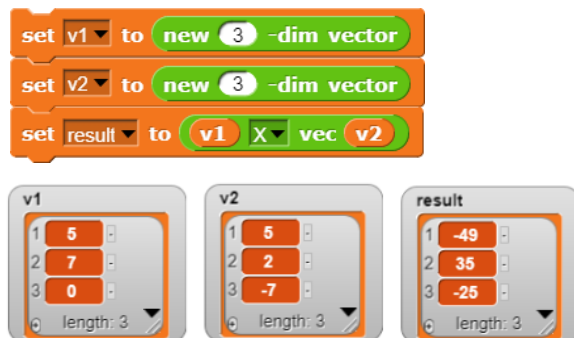
Operations between scalars, vectors and matrices are performed by the two blocks – if possible.

*<operand> <operator> vec <data>* bzw.

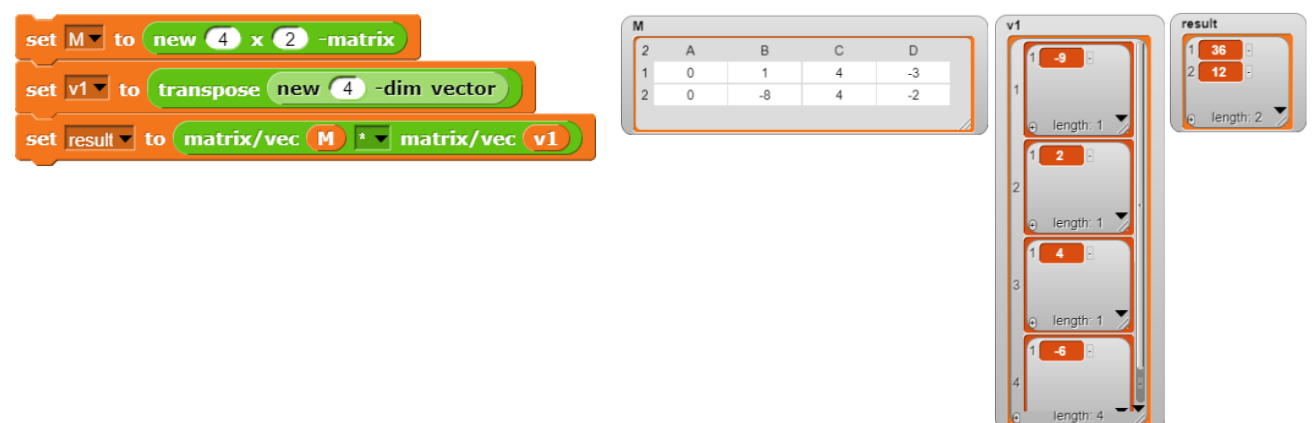
*matrix/vec <op1><operator>matrix/vec <op2>*



Example: cross product of two vectors

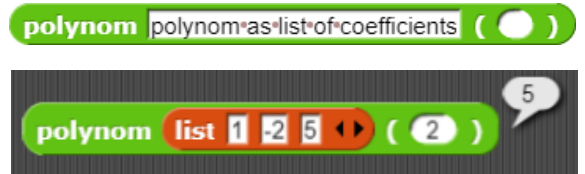


Example: product of matrix and vector



The following operators are more specific:

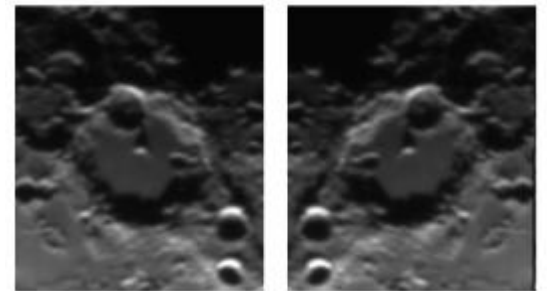
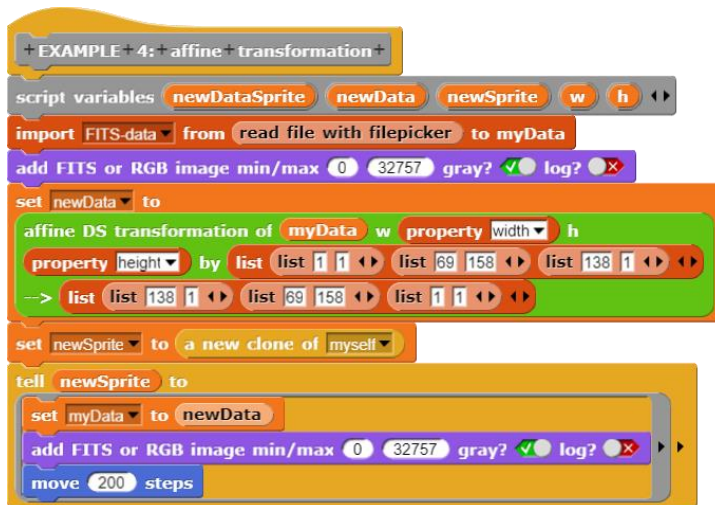
*polynom*<poly>(<value>) calculates the value of a polynomial for the specified value according to the Horner scheme, where a polynomial is described by a list of its coefficients, starting with the highest. The polynomial  $x^2 - 2 * x + 5$  would thus be represented by the list `list 1 -2 5`.



Affine transformations on images are performed using the block

*affine transformation of* <imagedata><width><height> *by* <points1>  $\rightarrow$  <points2>.

Example: Mirroring an image on the vertical



Linear systems of equations can be solved with *solve* <matrix>\*x=<vector>.



An application is polynomial interpolation

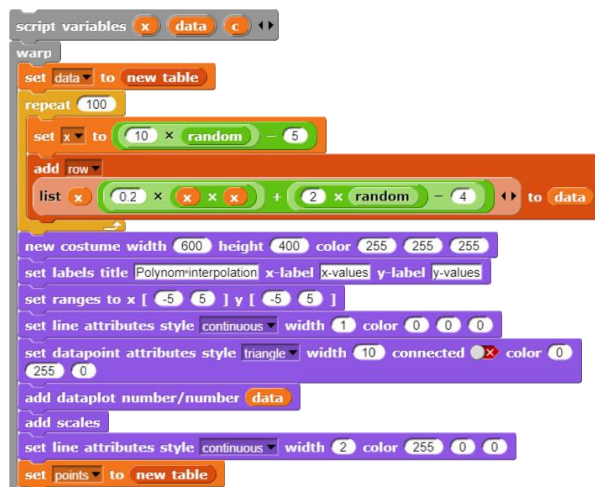
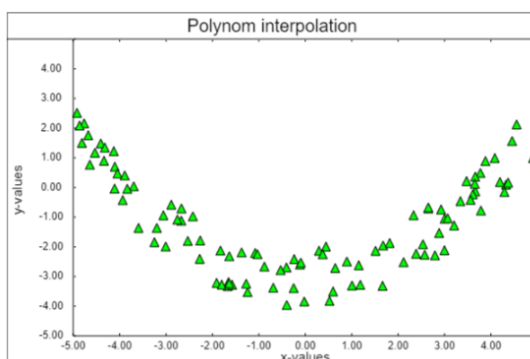
*polynomial interpolation for* <table>.



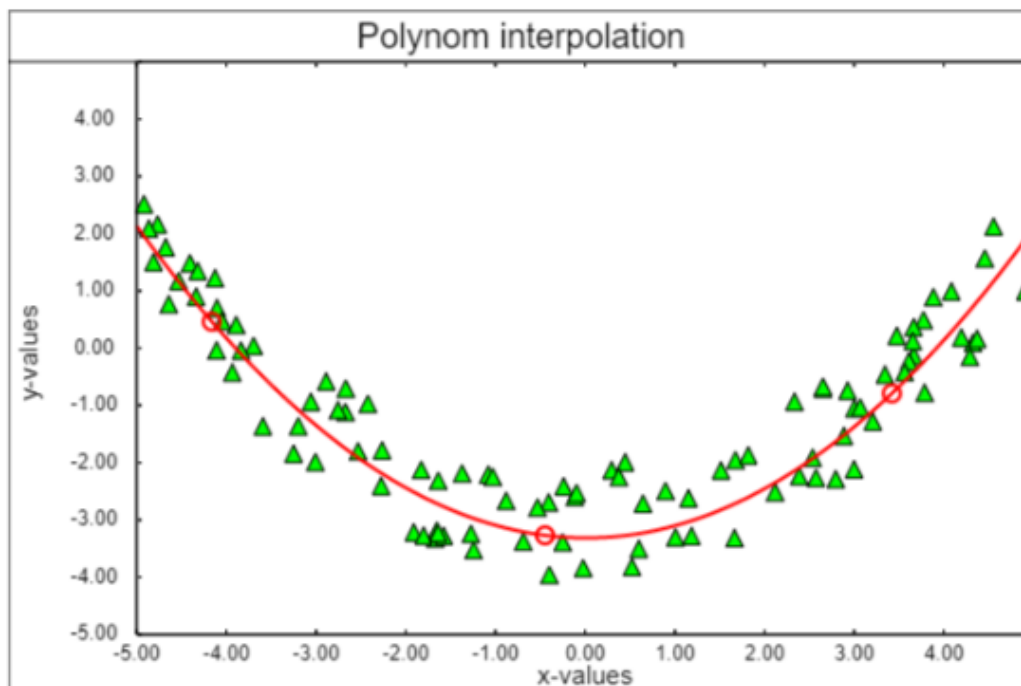
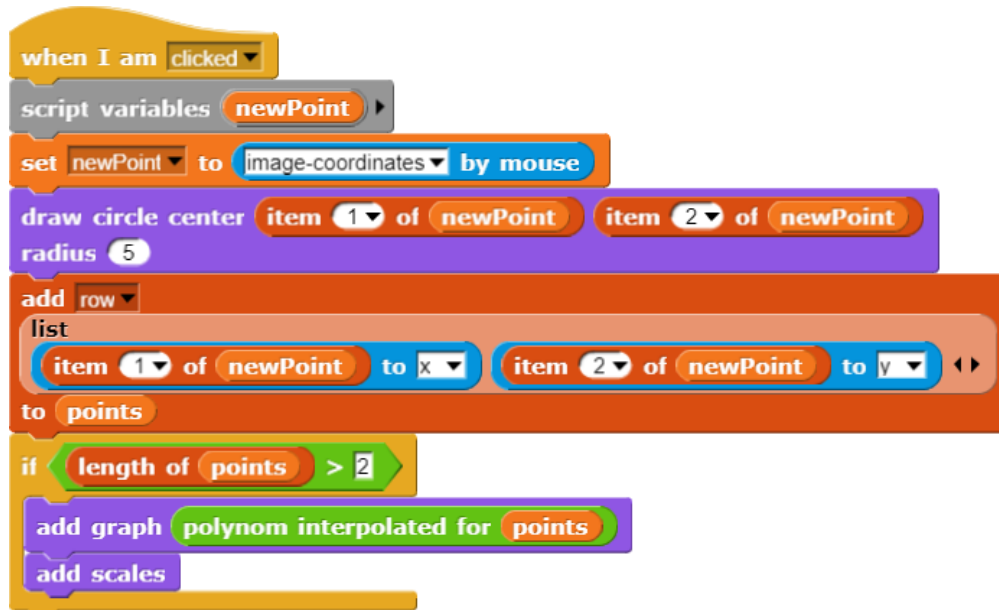
where a polynomial is calculated by n points.

Example: Curve through n points

We create some random points around a parabola and display the result.



Since we obviously can't get any further with a regression line, we click three points in this set of points. After the third point we calculate an interpolation polynomial and draw it in the diagram. This requires two coordinate transformations.



The last blocks of the operator palette round a number to the specified number of digits, which would be helpful, for example, when improving the diagram display, and provide random numbers between 0 and 1 in full length. The last block converts a list of texts into a string with given partial lengths. This allows, for example, the column headings of a table to be quickly converted into axis labels of a diagram.

round to 2 digits

random

--> label charwidth 7

## 5 Applications of the DataSprite

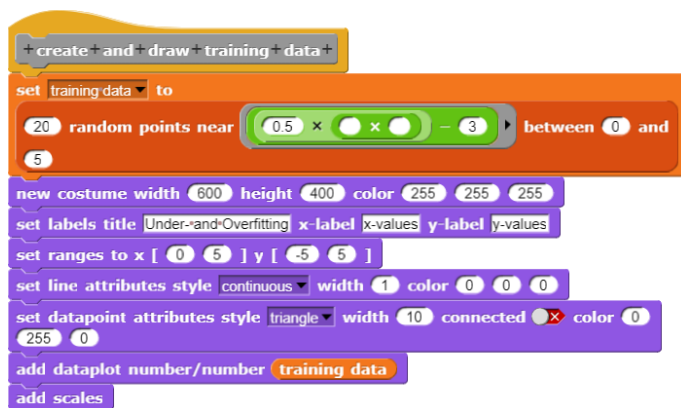
### 5.1 Under- and Overfitting

Machine learning uses training data to adjust the parameters of a function so that other values are well predicted - if all goes well. You build a forecasting tool, a kind of "telescope" for data.

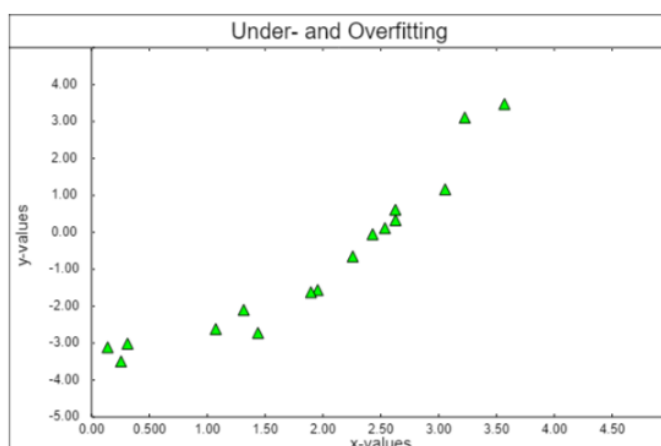
If the training data is reproduced well by the function, this does not mean that this also applies to other data. It depends very much on the type of function that is created. As application we choose the last example: the polynomial interpolation.

The task is: *Using training data, the coefficients of a polynomial are adjusted in such a way that OTHER data are predicted as well as possible.*

First of all we want to generate some data to calculate an interpolation polynomial. To avoid having to start over and over again, we write a reporter who calculates a table with  $n$  data of any function.

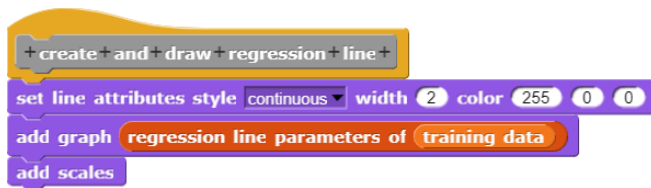


We use this to generate the training data, which we also display immediately.

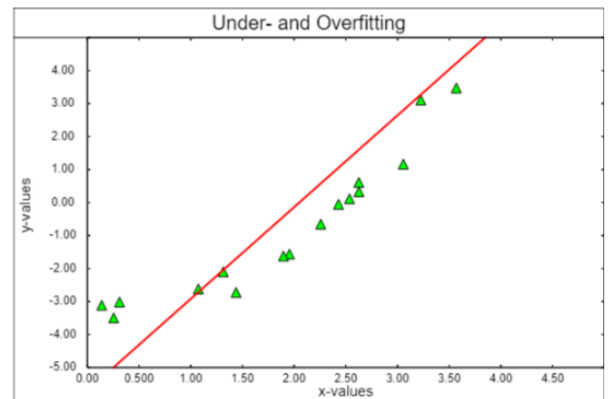




First of all we try a regression line.

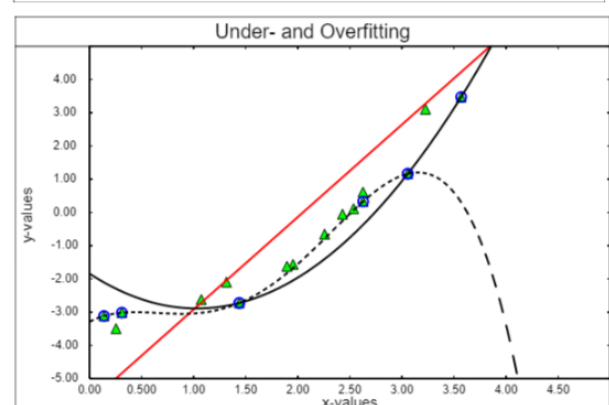
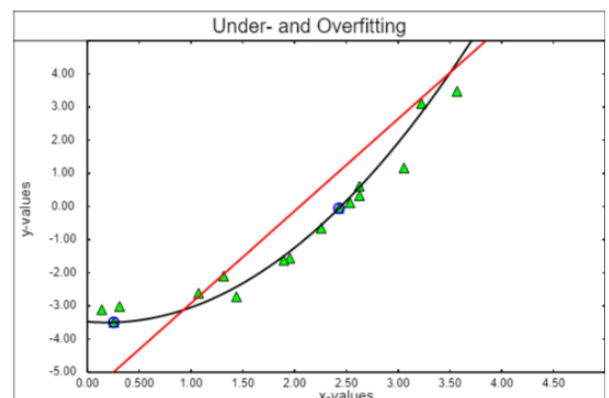
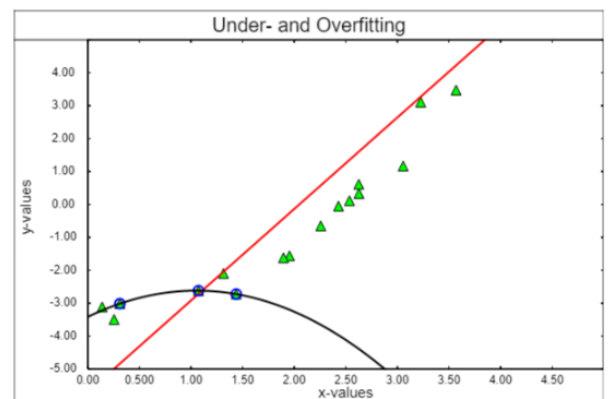


That actually looks quite nice, but on the sides it does not fit so well.



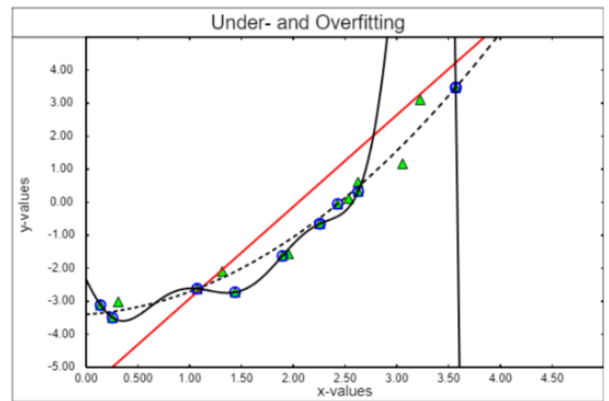
So, we'll try a polynomial interpolation.

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 by  $n$  points. We hope that everything goes well with the selection! The results depend on which points were wiped out. Enclosed a bad and a quite good result.



Now we're getting brave! Instead of three points we choose 5. After all, we want to do a good job! That works great up to the right edge, and then - opps!

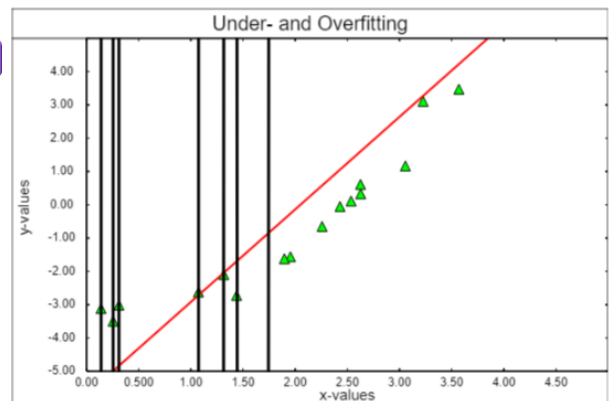
Maybe we just need to take more points. Let's try it with 10. The polynomial runs through more points, but at the edges it "runs away".



Well, then, with all the points!

```
set DS line attributes style continuous width 3 color 0 0 255
add DS graph polynom interpolated for training data
add DS scales
```

One can see that with increasing degree of the polynomial more training data lies directly on the graph, but that in between by the wild oscillations of the polynomial only senseless values are "predicted".



So, the quality of what we learn depends very much on how we deal with deviations. We have to decide which inaccuracies can be tolerated in detail so that the overall forecast is 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 to determine a "good" degree of 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.



## 5.2 New York Citibike Tripdata [NYcitibike]

Even in New York, cycling has become "hip" and borrowing data can be loaded as CSV files. We do so and load the almost 600,000 data records from June 30, 2013 into a table. We split the column headings to get a pure data table.

What did we actually find there?

The data legend provides the interpretation for the data: *Trip Duration (seconds), Start Time and Date, Stop Time and Date, Start Station Name, End Station Name, Station ID, Station Lat/Long, Bike ID, User Type (Customer = 24-hour pass or 3-day pass user; Subscriber = Annual Member), Gender (Zero=unknown; 1=male; 2=female), Year of Birth*

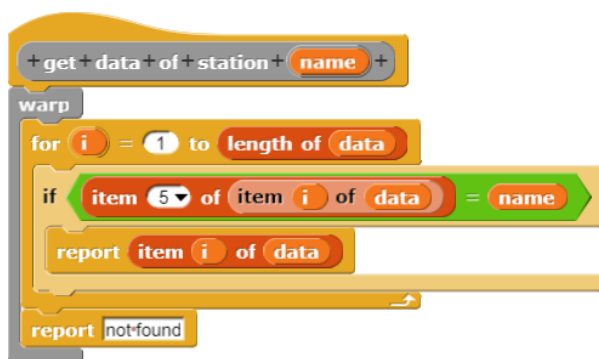
Since the geographical longitude and latitude of the rental stations are given, it is a good idea to use the *Word Map Library* from Snap!. We write a small block, which shows the surroundings of a rental station as a map.

Let's see where you can rent bicycles. For the overview we extract the rental stations from the complete list, e.g. by grouping them according to the name of the starting station (column 5) and selecting only this column as the result.

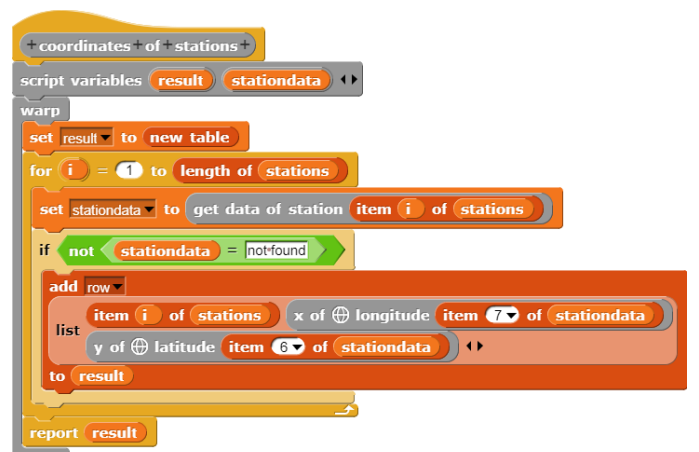
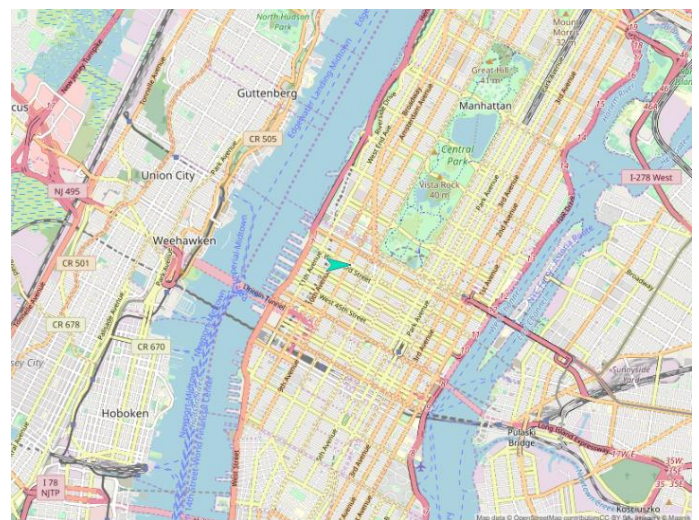
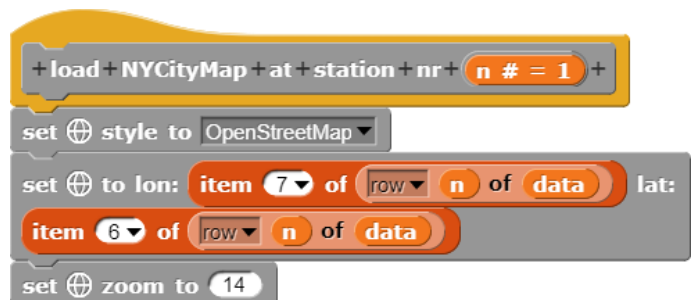


We get 337 stations after all.

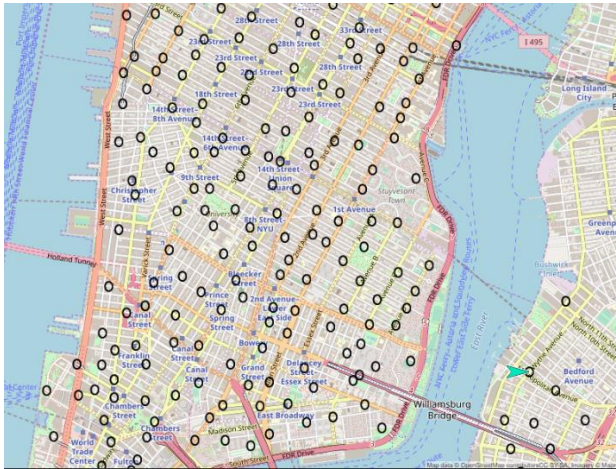
Then we collect the data of a station ...



... and build the coordinate list of the stations.



With these data we can send the sprite to the individual positions, where we leave circles with the *stamp*-block.



```

+show+all+citibike+stations+on+map+
warp
clear
set stations to
column 1 of mean of column 1 of data grouped by column 5
delete 1 of stations
set coordinates of stations to coordinates of stations
set size to 70 %
switch to costume circle
for i = 1 to length of coordinates of stations
go to x: item 2 of row i of coordinates of stations y:
item 3 of row i of coordinates of stations
stamp
set size to 100 %
switch to costume Turtle

```

At least in Midtown Manhattan, we don't have to worry about finding a rental station!

Now we want to have a closer look at the rental station Broadway - corner 41 Street (No. 55). To do this, we look for all records from the list that start or end at this station. That's 5005 events that day. Times are entered in this list together with the (same) 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".

```

set borrowing data to reduce time columns of borrowing data to hours

```

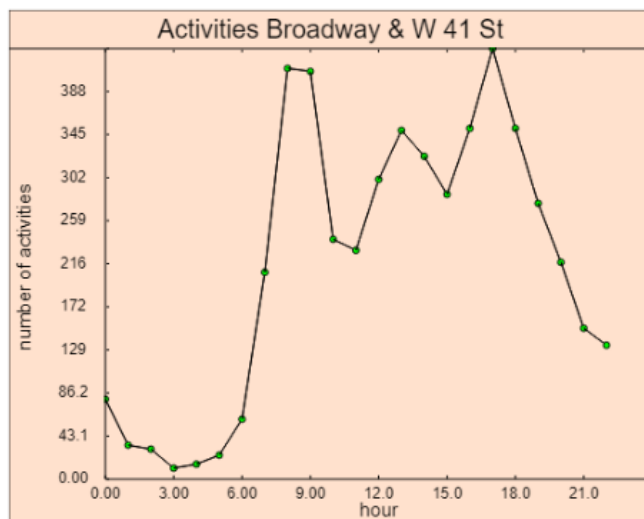
Now we can see what's going on at the station during the individual hours of the day.

```

set graph data to
number of column 1 of borrowing data grouped by column 2

```

And we can display this graphically as usual.



```

+connection+to+or+from+station+name+
report
keep items
item 5 of = name or item 9 of = name
set borrowing data to
connection to or from station item 55 of stations

```

```

+reduce+time+columns+of+table+to+hours+
script variables result
warp
set result to new table
add column column 1 of table to result
add column
map item 1 of split item 2 of split by by over
column 2 of table
to result
add column
map item 1 of split item 2 of split by by over
column 3 of table
to result
for i = 4 to 15
add column column i of table to result
report result

```

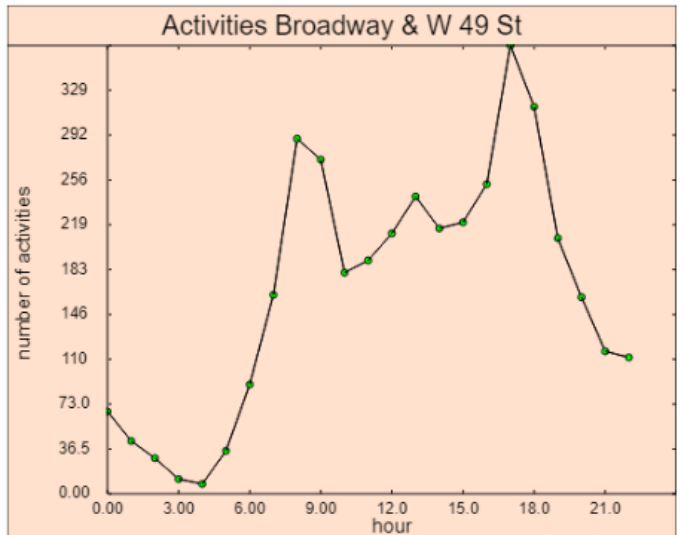
```

new costume width 500 height 400 color 255 225 205
import table-data from graph data to myData
set labels title join Activities item 55 of stations x-label hour
y-label numberofactivities
set datapoint attributes style o circle width 5 connected checked color 0
255 0
set scale attributes precision 3 textheight 12 number of intervals
x-axis 8
set ranges to x [ 0 24 ] y [ 0
max of column 2 of graph data
]
add dataplot number/number myData
add scales

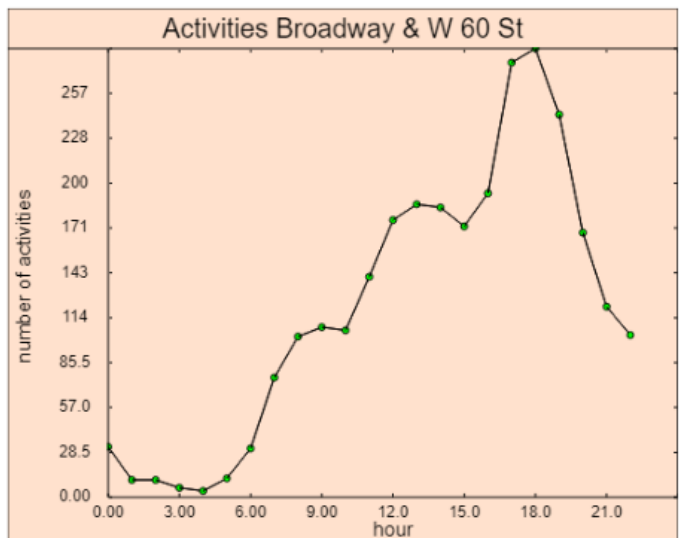
```

A few streets down the road, it looks similar.

Is that a general pattern?



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



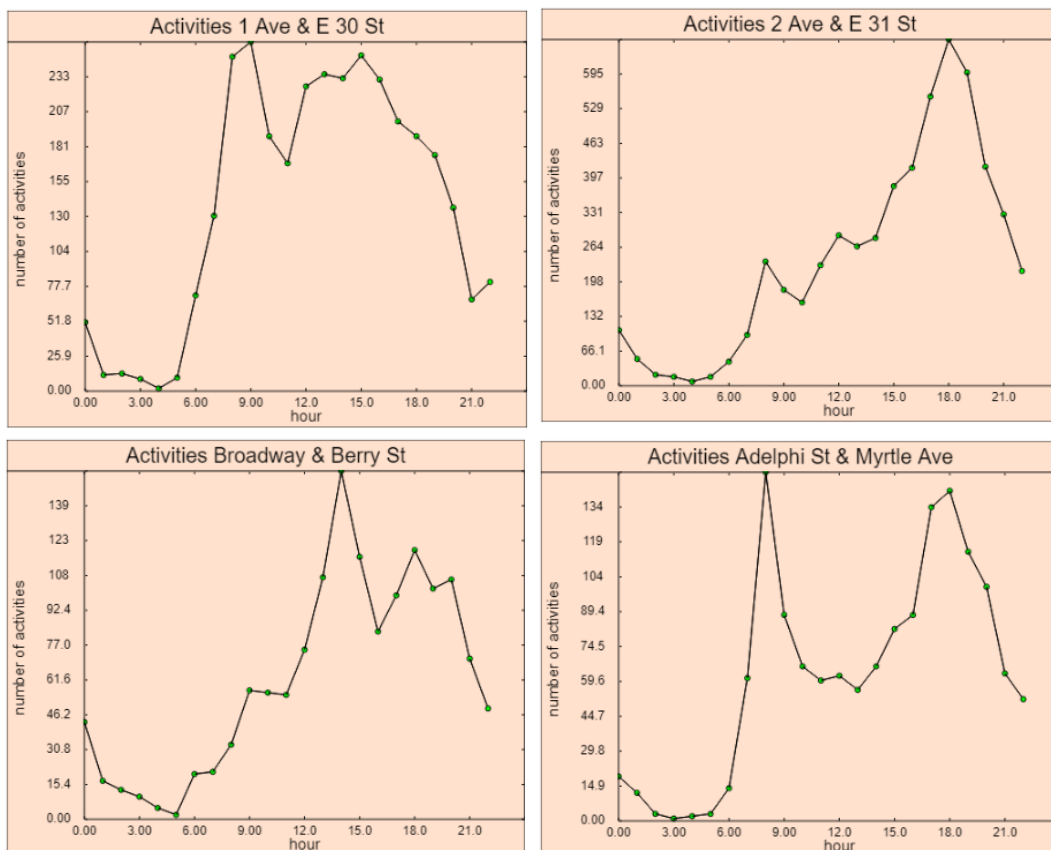
What can our programs learn from these data?

- We could, for example, predict from the usual departures and arrivals as well as from the actual stock whether sufficient bicycles will be returned in time at a station or whether it would be better to transport some of them there.
- We could determine which accu are needed for eBikes from the average path lengths.
- We could determine whether women or men would rather borrow the bikes at a certain time of day and then make sure that the offer is right. We could do the appropriate thing for the age of the borrower.
- We could determine the borrowing data per bike and predict when repairs will be due. We could also do this, for example, depending on the location of the stands.
- We could try to generalize distributions from some stations in such a way that forecasts for others can be derived from them. So, when the museums close at Central Park, the program can "learn" from the old data in which districts the bikes will presumably be delivered and warn if there are not enough free slots available.

etc.

Tasks:

1. Break down the activities of the stations according to arrivals and departures.
2. Write a forecast function that warns if there is a risk of a lack of wheels at a station in the next few hours.
3. For certain stations, display the connections to the most selected delivery stations graphically on the map using direct lines. Select the thickness of the lines according to the number of borrowing operations and the colors depending on the station. Are clusters formed?
4. Find out with the help of correlations-block whether there are correlations in rental behavior (e.g. with regard to times of day, location, ...) with gender, age, status of borrowers. You may have to replace the data with numeric data beforehand - similar to the times. Discuss possible consequences.
5. For a small section of Midtown (where everything is beautifully right-angled), find the coordinates of the street corners. Then develop a router that shows the shortest route to the nearest Citibike station.
6. The rental numbers depending on the time of day show quite a difference in different areas of Manhattan. Systematically examine similarities and differences and try to explain the results.

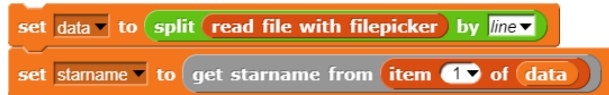




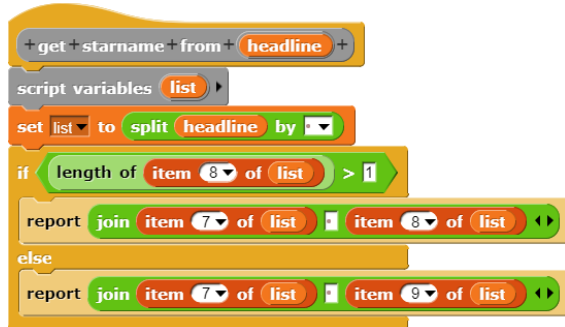
### 5.3 Star spectra [UniGOE]

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

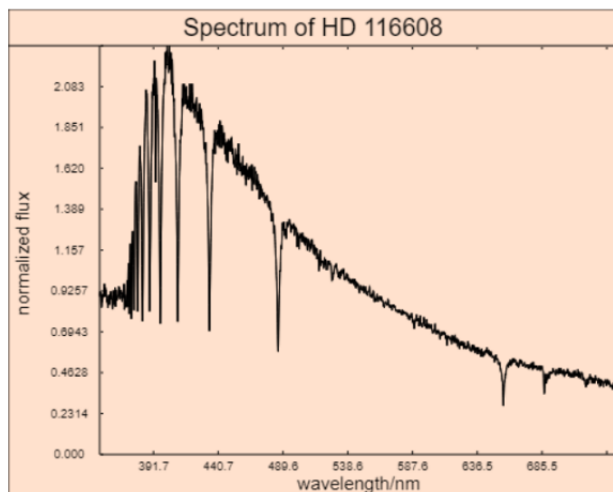
We get some star spectra (source: [UniGOE]) and save them as a text file. We read in such a file. In the first line we see the star name after the column captions. We isolate it and store it in the variable *starname*.



We know the star's name now. If you search the Internet for it, you will find a wealth of information about it. For repeating the loading process with other data, we encapsulate it in a separate block. After its execution, the actual star data are available as a table.



With these data you can quickly create a diagram.



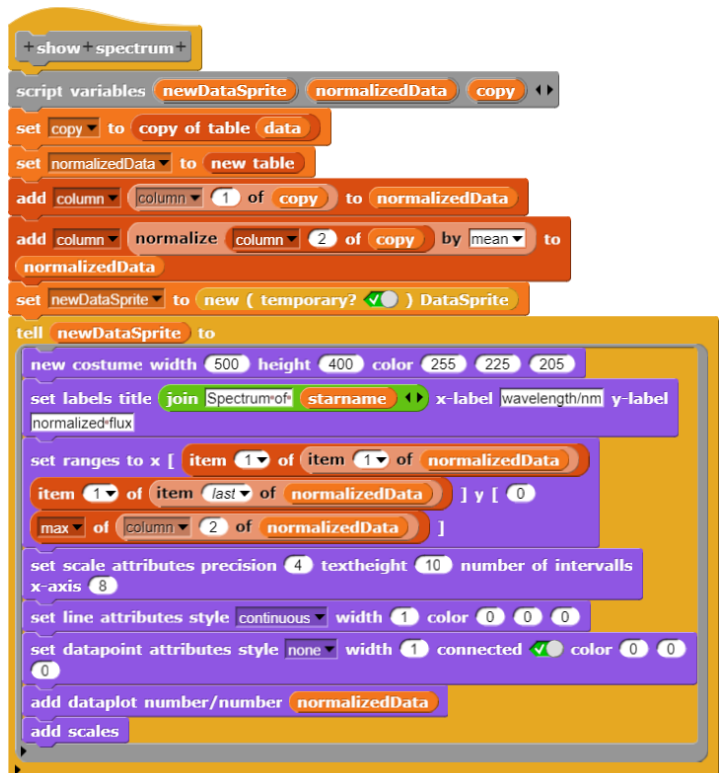
One recognizes well the falling course with some striking absorption lines. But does one need all spectral data for this insight?



```

# 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...
  
```

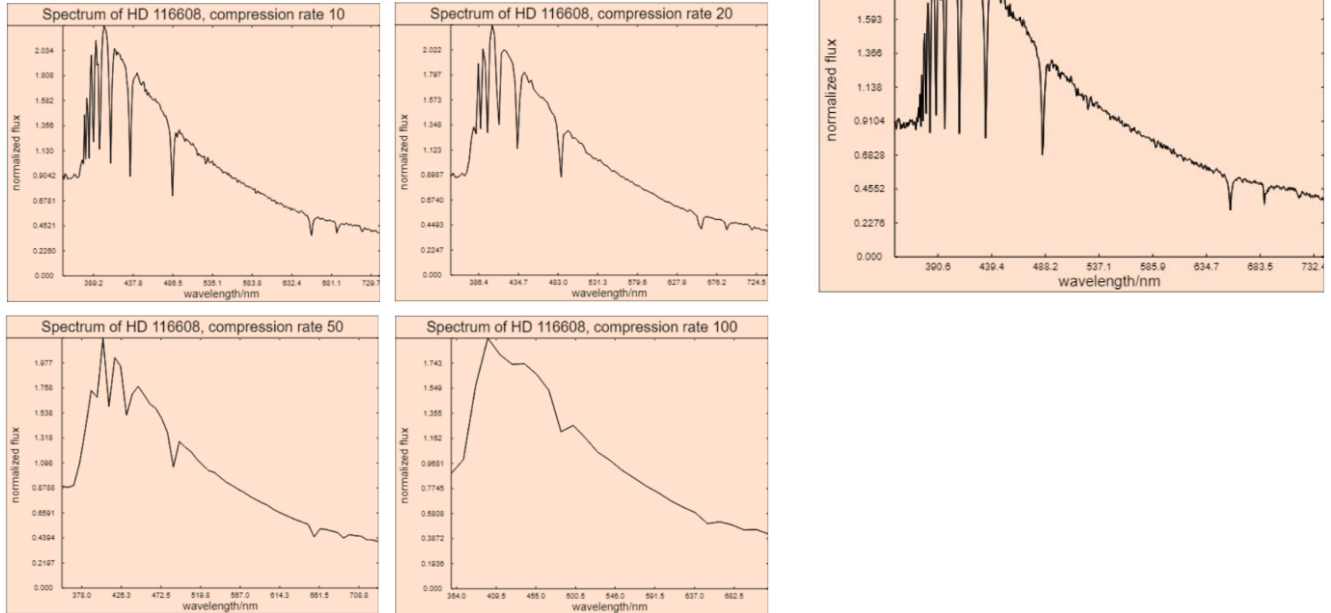
starname HD 116608



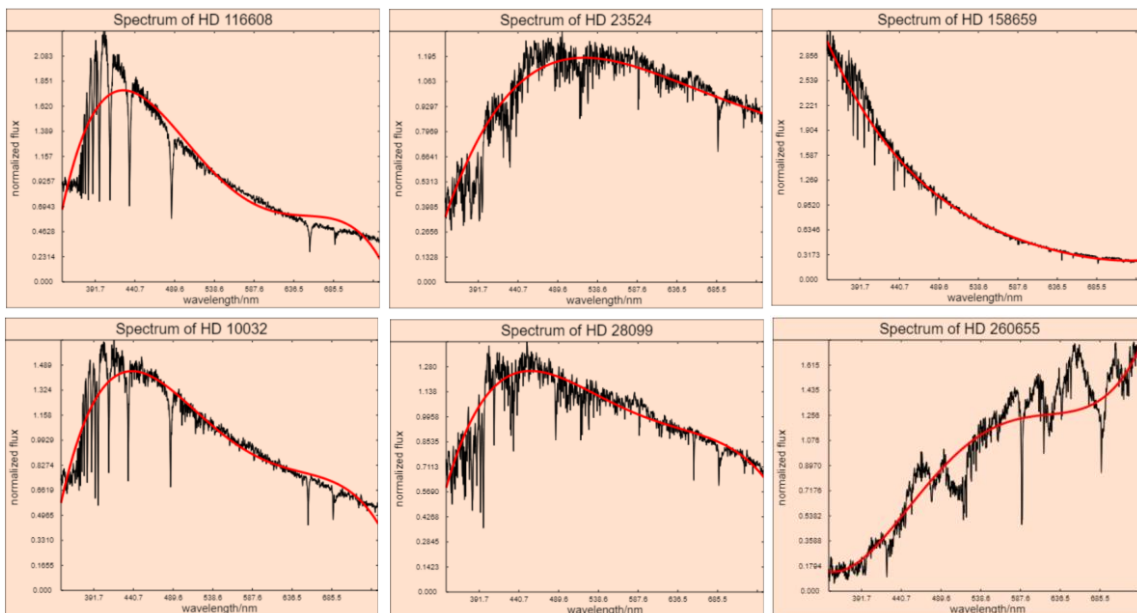
Perhaps it is sufficient to reduce the amount of data by averaging. We introduce a compression factor *compression rate* and add the script before creating the diagram.

set normalized data to  
compress data normalized data with factor compression rate by  
averaging

Factor 5 doesn't change much. So, let's try again.



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 e.g. 4th degree.



This also works perfectly! If we also record the polynomial parameters during the examination, 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	0.0000028988621087800814	-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	-0.00024815800544876965	0.11348374829256708	0.11348374829256708
7	HD 260655	6.248027476637483e-10	-0.000001337322548726115	0.0010450333683869723	-0.3486709605339992	-0.3486709605339992

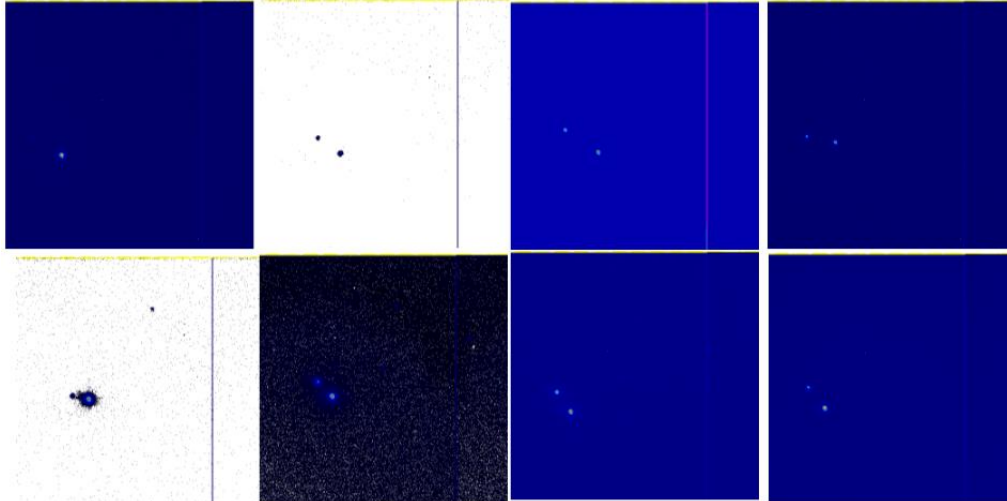
The program can "learn" which parameter intervals belong to which star classes on the basis of the old data. If you enter the data of a new star, it determines the coefficients of the polynomial and then gives a well-founded prognosis of what kind of star it might be.

Tasks:

1. Set up an interpolation polynomial of the lowest possible degree for the uncompressed spectrum data. Which points should be selected for this? Are there any differences between these polynomials and the results of the method shown above?
2. Develop a script that assigns an unknown spectrum to one of the previously occurring types.
3. Develop a method to examine the most prominent absorption lines more closely. Enlarge them for stars of the same class and try to determine differences "automatically". Discuss your ideas before realization.

## 5.4 Period of a Cepheid [HOU]

Cepheids are stars whose brightness fluctuates periodically. Since their luminosity depends directly on the period of the fluctuation, they serve as "standard candles" for distance measurement in space. We want to measure this period. For this we get some pictures of a Cepheid, which were taken on different days (Source: [HOU]). The example is used to explain *DataSprite* image operations.

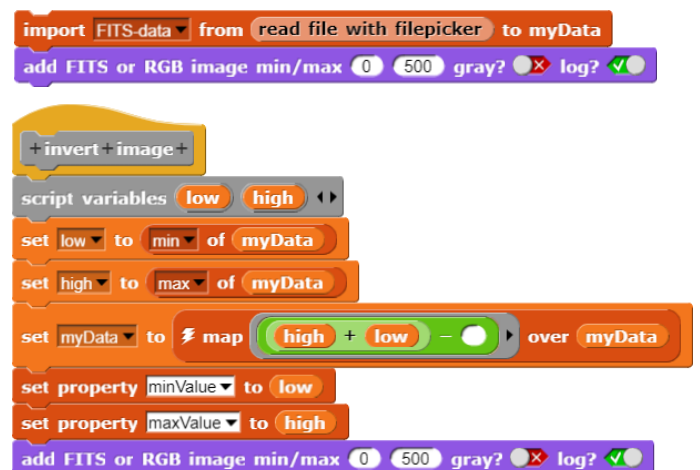
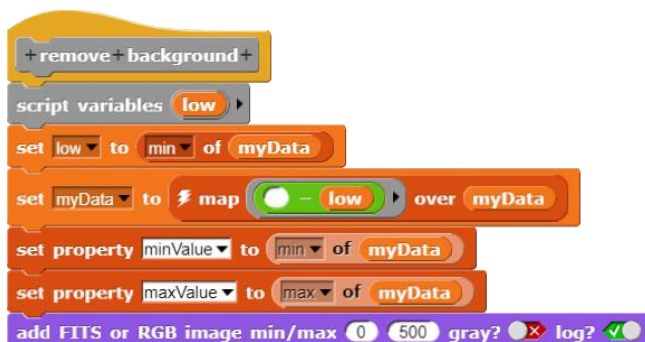


You can see that the pictures have very different quality. On some you can hardly see the Cepheid, the star next to the bigger star. Sometimes the background is brighter, sometimes darker. So, we have some tasks:

- Load the images, set a view that shows both stars well.
- Subtract the background radiation of the image, invert if necessary.
- Measure the brightness of a star.
- Display the results and find out the period.

And how do you determine the brightness of a Cepheid? It is calculated relative to an unchangeable star in the vicinity. So, in each image two brightness values have to be measured.

As in the other examples, an image is loaded into *DataSprite* and displayed. Minimum and maximum of the displayed values are determined experimentally in such a way that stars are clearly visible. (Remember: the actual image values are not changed by this.) If necessary, the image is inverted.

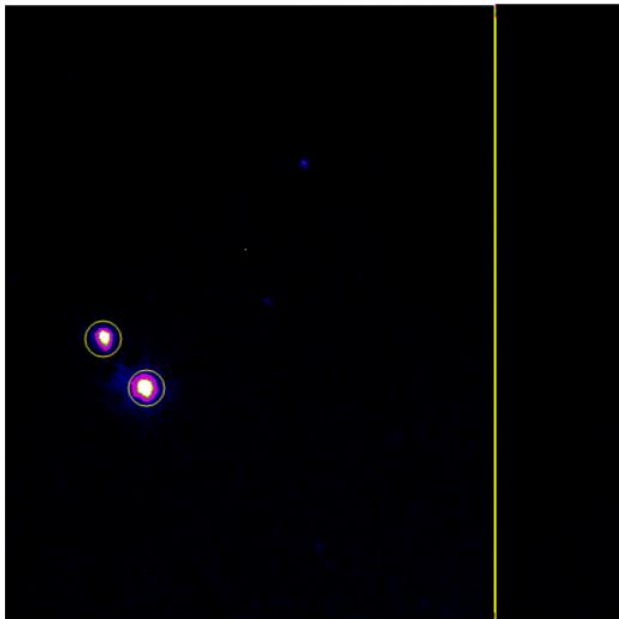




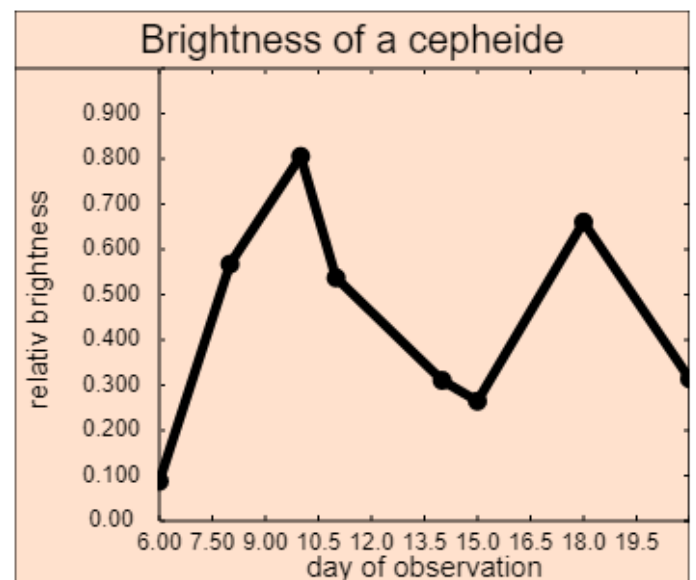
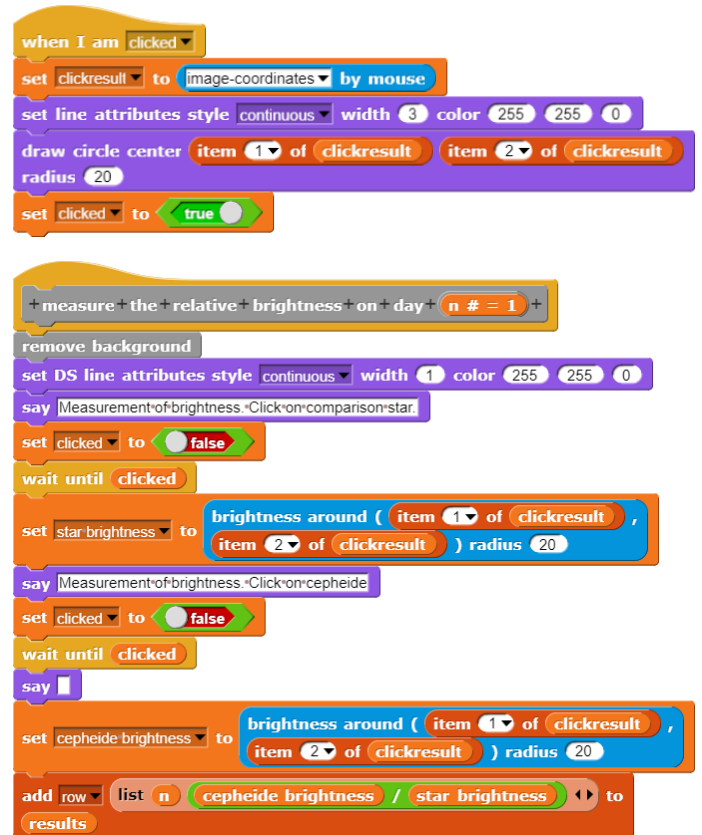
The radiation background is subtracted by simply subtracting the smallest image value from all of them.

We process the mouse clicks to determine the brightness by setting a variable *clicked* to *false*. If the image is clicked, it gets the value *true* and the clicked image coordinates are assigned to the variable *clickresult*. In the script, the program waits until data is available using the **wait until clicked** command.

With these aids, the measurement process can be summarized in a separate block. The results are assigned to a *result* variable. This will be evaluated later.



8	A	B
1	6	0.0876221753506
2	8	0.5674336949167
3	10	0.8054445068099
4	11	0.5370600093037
5	14	0.3104731277173
6	15	0.2646507209664
7	18	0.6603668797582
8	21	0.3140171254899



The period will therefore be about 9 days.

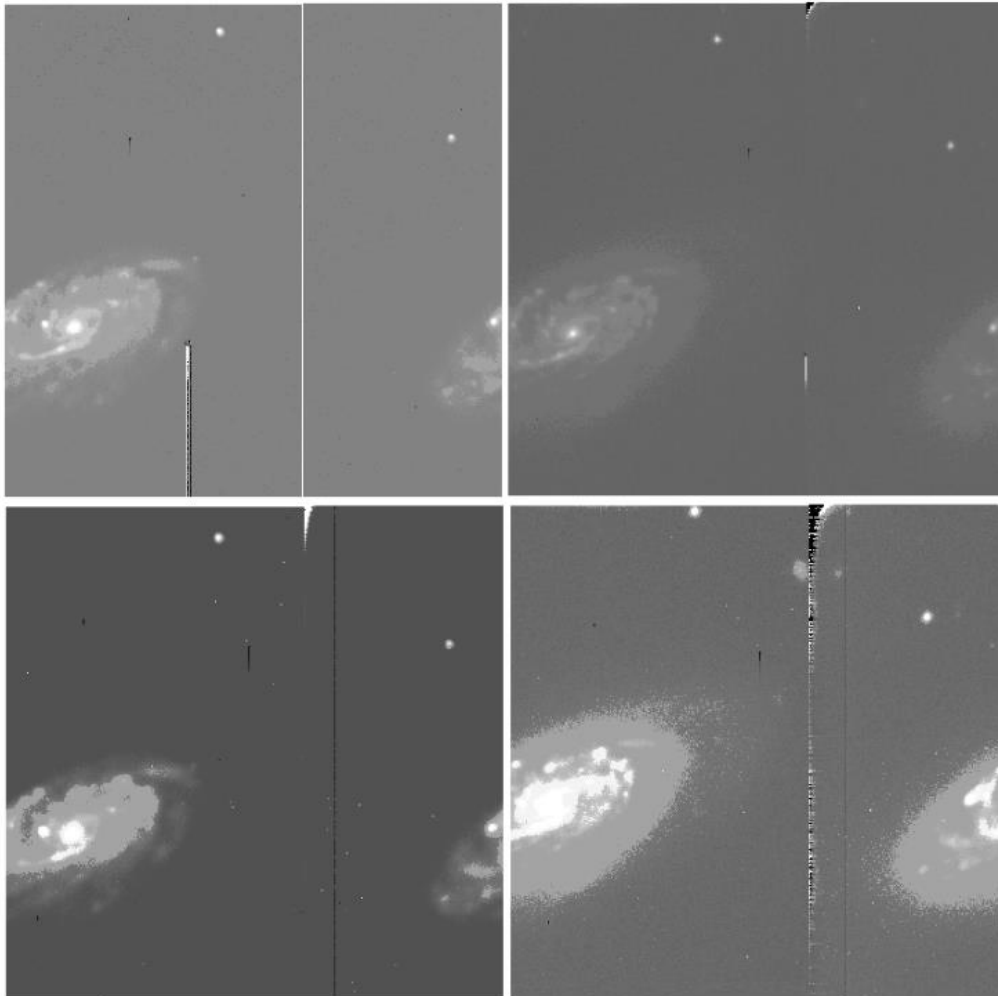
Tasks:

1. Place a nice smooth sine wave (or similar) through the data points. Experiment a bit. Does this make the period more accurate?
2. Develop a script that allows you to draw lines on the image that should correspond to the period duration. The period duration is then determined from the mean length of these lines.
3. On some pictures the Cepheid is hardly to be seen. Develop a script that shows where stars are in the image, perhaps by a cross. What is a (pictured) star anyway? The vertical line on the pictures is probably not!

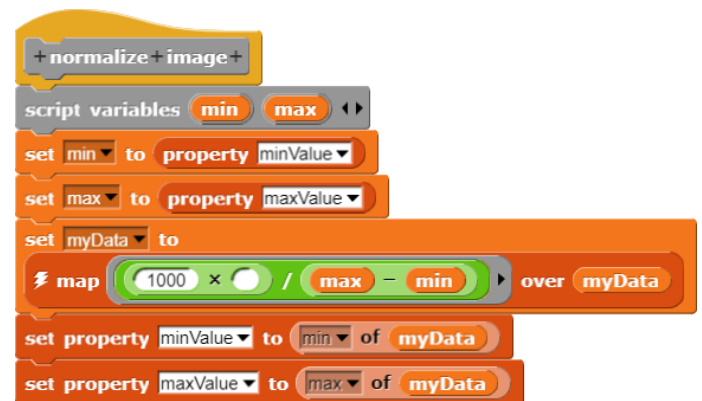
## 5.5 Search for a Supernova [HOU]

A new supernova is not so easy to discover between the many other stars. But you can identify candidates by subtracting old images of a sky from new ones. If a bright (or dark) spot remains, you should take a closer look at this area (source: [HOU]). The example again explains the use of *DataSprite* image operations.

So we get ourselves four pictures taken on different days, of course with different exposures, different background radiation, ... and also some other quirks.

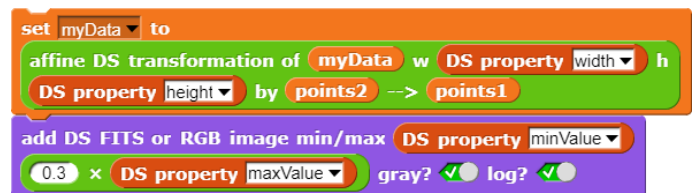
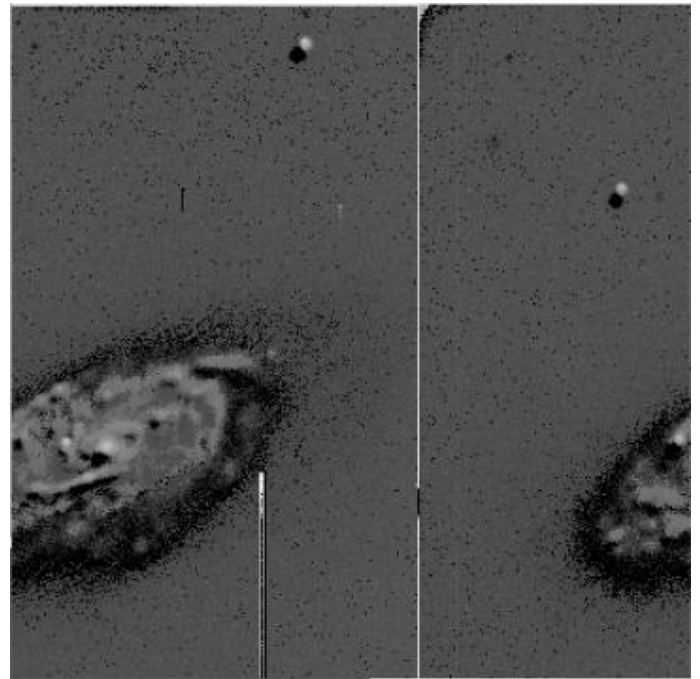
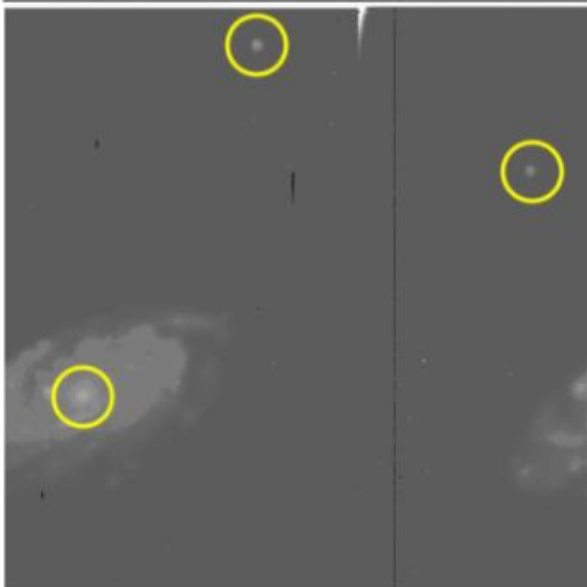
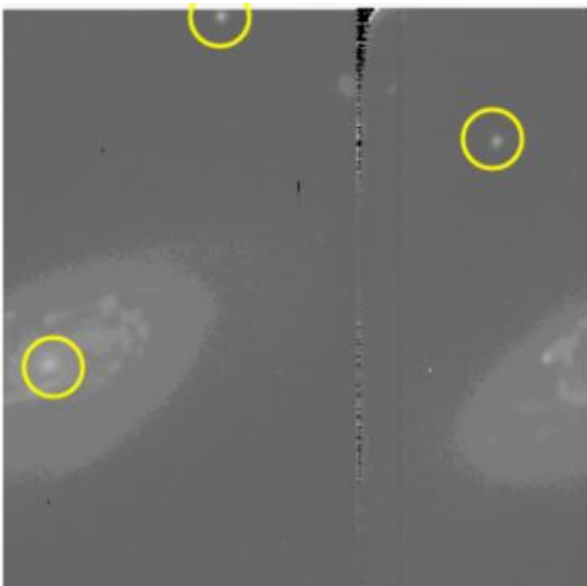


In order to see anything at all on the pictures, they were first "normalized", i.e. the background was subtracted and the rest of the picture values were mapped to the range from 0 to 1000. But this does not give them the same overall brightness, as the result depends strongly on the highest image values.

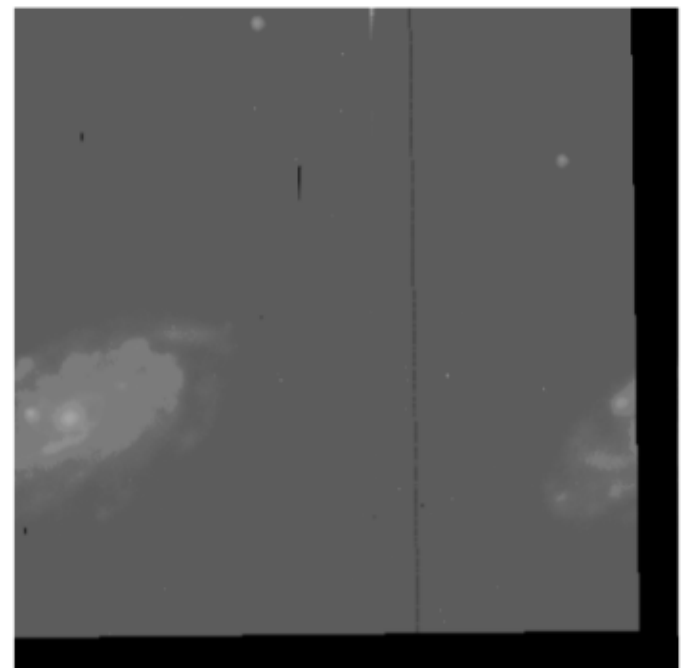


If we now simply subtract one image from the other, then we get a three-dimensional result with some "Ying-Yang effects" where bright image areas did not fit together completely. We can use the vector arithmetic block for this operation because FITS data are a simple list.

Therefore we carry out an affine transformation with one image by clicking and saving three corresponding points first on one image, then on the other.



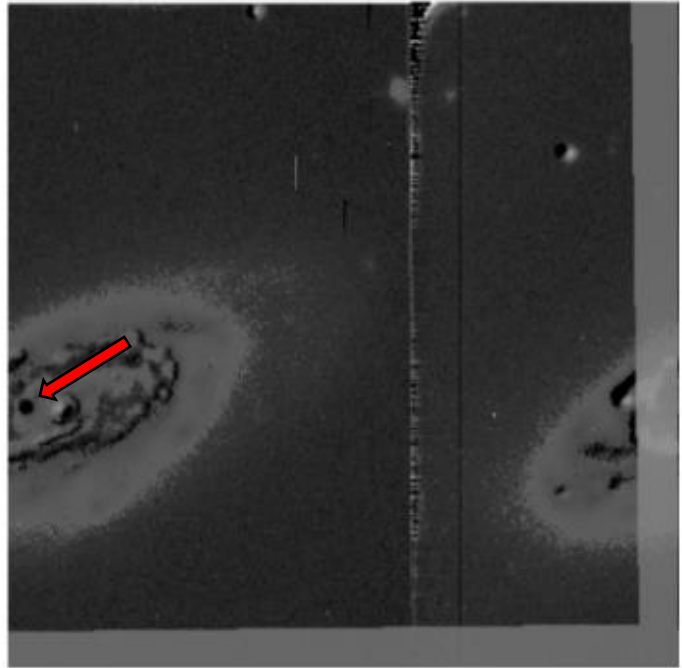
The result:



We'll subtract this picture from the first.

Well, it goes better too!

But at least we found a new shining spot in the galaxy! (It's black here because we took the picture with the supernova candidate last. But we didn't know that before. 😊)



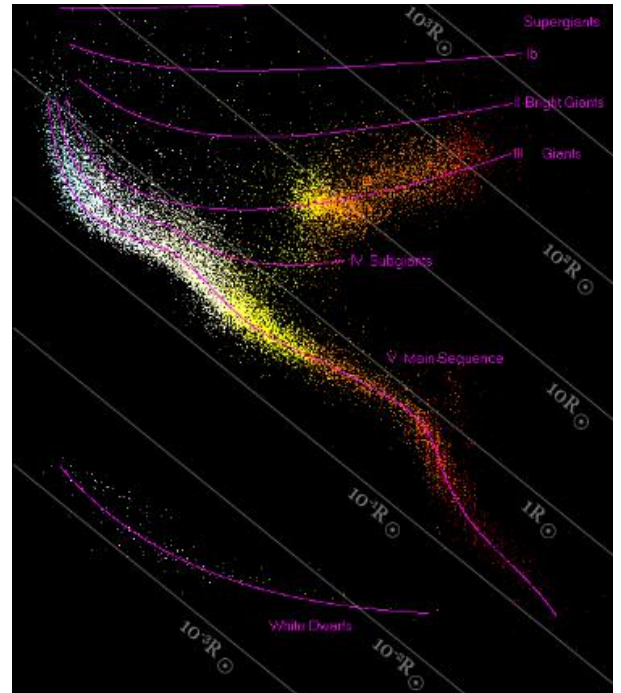
Tasks:

1. Find images of the same sky area with and without Nova and process them as shown.
2. Automate parts of the search as much as possible. Discuss the difficulties.
3. Can you improve the accuracy of the "handiwork" in supernova search, e.g. by focusing the clicked stars better? Try it!

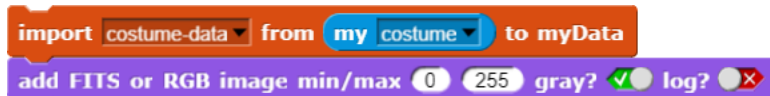
## 5.6 Classification of stars according to the kNN method

In the Hertzsprung-Russel diagram (see Wikipedia) the luminosity of stars is plotted above their star class. The result is a kind of line from top left to bottom right, the "main sequence". On this line stars like the sun are mostly located. Right above the main row we find the red giants, left below the main row the white dwarfs. That's enough for first. (Picture source: [HR])

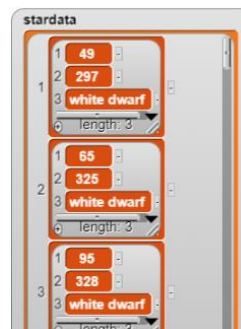
We want to classify new stars in this diagram using the k-next neighbor (kNN) method: As training data we generate a list of stars with their coordinates (simply as image coordinates in the diagram) and their type. If we want to classify a new star, we determine its position in the diagram and look for 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 a picture of the Hertzsprung-Russel diagram ([HR]). We import it into Snap! as a costume and generate the required data from it.



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



Then we can classify new stars by clicking on them (here) and labeling them.



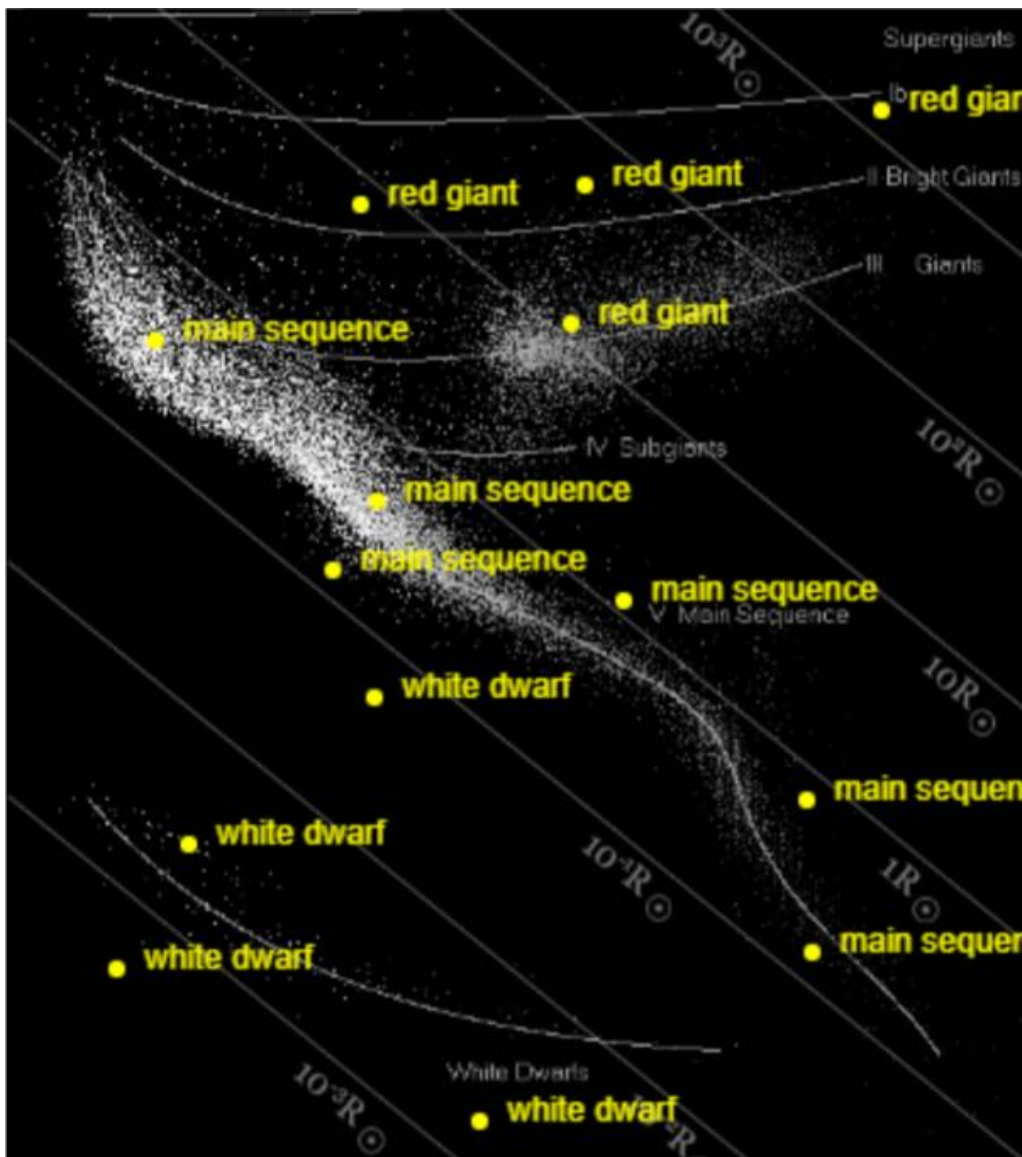
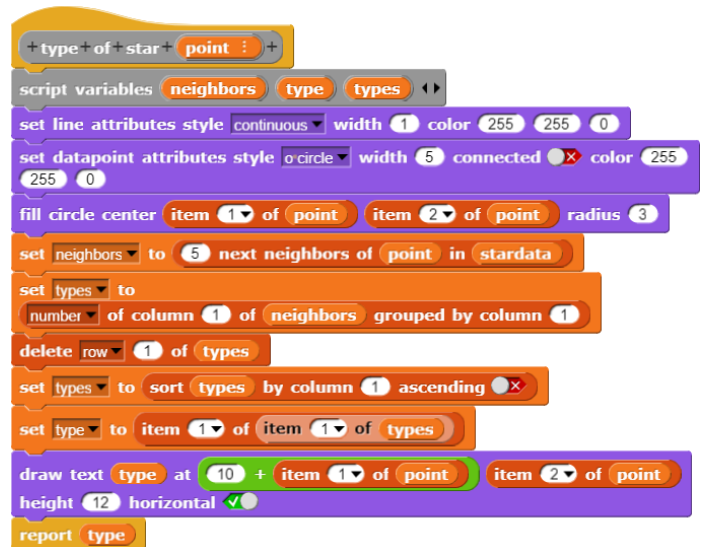


We set some properties for the representation ...

...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. In the result we delete the headings and sort the list in descending order. The type of the new star is then the first element in the first line. We write this next to the star.

The result:



---

## 6 Hints

The examples were largely chosen from astrophysics because the *DataSpriteLibrary* will be used in this area in the next semester. But the library operations are of course not limited to this area.

Machine learning consists to a large extent of the preparation of data - regardless of whether it is table data or images. The actual learning processes of the machines then consist of the parameter adjustments resulting from the data. Since both can be easily visualized, there is a broad field for beginning programmers with many transitions to the field of "informatics and society".

Examples of how to use the operations of the *DataSpriteLibrary*, especially the convolution using a kernel, can be found in [DBV].



---

## References and sources

- [Albon] Albon, Chris: Machine Learning Kochbuch, O'Reilly, 2019
- [Baumann] Baumann, Rüdiger: Didaktik der Informatik, Klett, 1990
- [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](https://de.wikipedia.org/wiki/Flexible_Image_Transport_System)
- [Goodfellow] Goodfellow, I.; Bengio, Y.; Courville, A.: Deep Learning, MIT Press, 2016
- [Grus] Grus, Joel: Einführung in Data Science, O'Reilly, 2016
- [HOU] Hands-On Universe: [handsonuniverse.org/](https://handsonuniverse.org/)
- [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>
- [Minsky] Minsky, Marvin: Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, New York, 1967
- [Modrow1] Modrow, Eckart: Technische Informatik mit Delphi, emu-online, 2004
- [Modrow2] Modrow, Eckart: Zur Didaktik des Informatikunterrichts – Band 2, Dümmler, 1992
- [Rojas] Rojas, Raúl: Neural Networks - A Systematic Introduction, Springer Berlin, 1996
- [SAP] [www.sap.com/germany/products/leonardo/machine-learning.html](https://www.sap.com/germany/products/leonardo/machine-learning.html)
- [SQL] Modrow, Eckart: Informatik mit Snap!, <http://ddi-mod.uni-goettingen.de/InformatikMitSnap.pdf>
- [SZ] Sueddeutsche Zeitung: 10. April 2019  
[www.sueddeutsche.de/wissen/schwarzes-loch-bild-1.4404130](https://www.sueddeutsche.de/wissen/schwarzes-loch-bild-1.4404130)
- [UniGOE] Institut für Astrophysik, Universität Göttingen