o++o
Tabments - Queries, Calculations, Statistics and Visualization

Klaus Benecke

(24.10.2023)

Foreword

What does it mean that o++o (ottoPS) is probably the simplest programming language?

It does not mean that o++o consists of very simple concepts. It does mean, however, that its application is relatively simple. o++o is not simple, but solving problems is easier than with other programming languages of equal expressiveness. o++o behaves like a natural language. English or German is not easy to learn either. However, natural language can be used - to a certain extent - even by children under the age of four.

We are convinced that the basic idea behind our best operation (stroke list operation) is easier to understand than the multiplication algorithm of decimal numbers. In our opinion, the concepts of o++o are relatively difficult to formalize, but they can often be described by simple algorithms that almost every user (= OttoNormalVerbraucher) can use in the future.

Is o++o a programming language?

o++oPS is designed as an end-user language, but not for programming complex database systems or compilers. It was developed to support people in solving their mathematical everyday requirements. Daily challenges are first of all (ad hoc) queries to tables (databases), documents or collections of tables and documents. It also includes financial calculations or in other daily context: determination of function values, determination of zeros or extrema of functions and solving a system of equations (calculation with matrices). In addition, o++o should be able to generate and manipulate images and visualize tables and documents in the form of diagrams. The most important innovative ideas of o++o compared to other approaches are connected with repeating groups. This means that a given object may contain not only null or one value for an attribute, but also multiple values. For such structures, known for more than 50 years in computer science, o++o provides new, powerful and easy to use operations.

This book contains a variety of sample queries to illustrate the basic concepts.

1 Table of content

List of programs and queries

Introduction

The first computer language has already been developed during the Second World War by Konrad Zuse. Very early languages like Fortran and Lisp have been used for more than 60 years. BASIC was the first attempt to develop a computer language simple enough to be used by everyone. One of the original goals of SQL was to develop an end-user language. There are now well over 100 computer languages. Python - the easy-to-use programming language - and the other general-purpose languages are designed for programmers. We follow the conviction; it is advantageous if every person can understand and write computer programs.

Let us direct our view to Germany: It is to be noted that up to now there is no generally recognized computer language which is taught or can be taught in schools for all. Spreadsheets even appear in math books of all students, but EXCEL is not a computer language. With o++o, any student can write certain 5-line programs where EXCEL requires 7 worksheets.

Therefore, to this day, there seems to be no universally accepted computer language worth teaching to everyone, or at least user-friendly and powerful enough for its usefulness to be sufficiently visible even in school education. The only widely used "languages" are associated with the use of search engines. Here the user only has to write one, two or three words (a very simple program) and the system finds thousands, millions or even more results. However, in this context, the user is not able to write "programs" with more selectivity. He has to hope that Baidu, Google and Co. will find exactly the pages that meet his requirements. Extracts of these pages are written at the beginning of an almost infinite result set. What does the user do if he is more interested in a document with the rank number 100, 1'000 or 100'000? What can a user do, if he has an exact idea of 1'000 desired documents he is interested in, but he wants only some small sub-documents or sub-tables of each of them? These questions are not easy to answer and realize. The language o++o (otto) aims to solve parts of such problems. We summarize the main design principles and requirements for an end-user computer language or data model with corresponding operations:

1. It should be based on easily applicable concepts with a simple syntax.
2. It should be expressive and powerful.
3. It should be expandable with new operations.
4. It should have precise semantics based on algorithms.
5. It should allow queries on tables (databases) and documents.
6. It should allow queries over document collections (IR systems) and entire databases.
7. It should allow computations by naive (brute force) algorithms.
8. It should also be usable for people with little interest in mathematics and computer science (programming by gut feeling).
9. It is intended to provide simple as well as more sophisticated concepts for broad classes of applications, suitable even for users with a keen interest in mathematics and computer science.
10. It should solidly integrate single data and bulk data operations.
11. The result of a mass data operation should be as small as possible (e.g., no Cartesian product if possible; highly selective conditions must be present).
12. It would be nice if it could use graphical features based on structured tables.
13. It should be efficiently implementable.
14. At least parts of the language should be able to be optimized.

o++o was designed and developed with these principles in mind. It started as a database language for tables with repeating groups. A record with repeating groups may contain not only one value at each position, but also several (sub tuples of) values. For example, a student record may contain a name and a scholarship, but it may also contain multiple hobbies or multiple (SUBJECT,MARK) pairs.

Similarly, a machine part may contain a number and the name of a color, and to that several subparts or several layers or edges. Such sub-tuples may have sub-tuples again. These repeating groups have existed in computer science for more than 60 years. They are typical for hierarchical systems (IMS,...), but were later discredited by emerging relational systems. Even today they are widely used in XML, JSON and NoSQL systems. However, in our opinion, there is no widely accepted computer language capable of adequately handling these richer structures. With the advent of XML, we have been able to generalize our operations to the new capabilities of arbitrary tagging and the alternate operator (|). Therefore, we are able to manipulate not only tables, but also documents. We have introduced the name tabment. A tabment can be understood as an abstract (syntax-independent) specification of an XML document. Step by step we improved our language o++o. We introduced binary search trees in tabments. Thus, we have achieved great efficiency gains for several operations.

Indices can also be considered as tabments.

Our language o++o is implemented in OCaml. Some basic keywords of o++o are German or French ('gib' instead of SELECT, 'avec' instead of WHERE,...) because they are shorter than the corresponding English words, but most keywords are English. This seems to be important because smartphones have only a small screen.

When we consider certain queries, our o++o data model relates to the relational model perhaps like decimal numbers relate to the Roman system. Roman numerals are more understandable, especially for small numbers, but calculations are usually more difficult.

The most common argument against an end-user language is: Is the customer paying for a product for which he still has to learn something?

We think back 100 years:

The car was invented, but most people took the railroad or a horse-drawn carriage if they were rich enough. It was believed that there would never be more than 1 million cars in the world because there would be no more than 1 million drivers. No one believed that the average person would one day get a driver's license and drive a car himself.

Today, many people also believe that no one would buy a computer program that might take a few hours or days to learn.

We put forward the following arguments against it:

0. Almost all people in the world had to learn for several years to understand the single data operations addition, multiplication, division and difference for each number range in school. Are bulk data operations like selections, calculations, restructuring, sorting tables, ... not just as important?
1. Even if you want to use a word processor like MS Word, you pay for it and need weeks and months until you master all the possibilities.
2. A good programming language is much easier to learn than German or a foreign language. Knowledge of a computer language could become part of general education in the future.
3. A good programming language allows many problems to be formulated more briefly and precisely with fewer misunderstandings than any natural language.
4. There is no need to explain the advantages of someone who has a driver's license or even a car. If he can even program solutions to problems with the computer himself, this increases the quality of computer use, because he can also interpret the results better. He does not need a computer scientist (chauffeur). Thus, there are fewer communication problems and he saves the cost of the computer scientist and the time for communication.
5. If the individual can make precise queries, he has much more compact query results and saves a lot of manual search effort. This also reduces the workload and improves quality.

What are the more specific design principles of o++o?

1. **important things first**

1.1 Sorting by the first attributes of a collection

```
gib DEPARTMENT,CHIEF,(NAME,LOCATION m) m
```

Here is described a structured table, which contains for each department also a corresponding group of employees. Sets (m) (and multi sets) are always sorted by the first column names. I.e. the outer set is sorted by DEPARTMENT and the inner set by NAME and then by LOCATION, because the NAME is not always a key in a department.

1.2 First written - first calculated:

`2+3*4` gives `20`

Here, a rectangle has one longer side, which consists of two sections 2 and 3 meters long. The other side is 4 meters long. The area gives 20.

`3*4+2` gives `14`

If I have two rectangles, one with side lengths 3 and 4 and one with area 2, I can first calculate 3*4 and then add 2 to get the area.

1.3 TT-Invariance (TT=TabmentType)

For many operations such as addition or multiplication, the type of the result is the same as the type of the first input value.

```
<TABH!
SUBJECT, MARKl m
Math     1 2 4 1
Phy      2 3 5 2 4
!TABH>
*15/6
```

Here a whole table in horizontal tab format is multiplied by a number. That is, each number of the table is multiplied by 15/6 and the words remain unchanged. This results again is a table of the type SUBJECT,MARKl m. The renaming of MARK to POINTS can be done by the user. I.e. also here the first input value is more important than the second.

1.4 Exponent representation of numbers

o++o additionally allows a representation where the more important part - the exponent - precedes the mantissa of a number. The exponent says more about the size of the number than the mantissa:

6m12.345'678 (12 million 345 thousand ...)

9m123.456 (123 billion 456 million ...)

2. pragmatics and methodology first

We can also allow multi-line semantics for a single term. Then we could

(23+45+67) * (1111+2222+3333+4444)

through

```
23+45+67
*
1111+2222+3333+4444
```

replace. This can be typed faster and is also clearer by dedicating a line to each pair of parentheses. In o++o this notation is further shortened to

```
23+45+67
* 1111+2222+3333+4444
```

This is not done for methodological reasons (better readability), but for pragmatism. This notation does not waste the additional middle line. Compared to the first notation, you have to use a (larger) return key only once instead of 4 brackets.

3. short catchy keywords

Short programs require short keywords and short operation symbols or names. However, if the number of these symbols becomes too large, one must also allow full names for designations so that the user can remember them. For o++o, the more important a symbol is, i.e. the more frequently it is used, the shorter it is. This rule can be better implemented by allowing non-English words as well.

Very short are + , * ,... m, l ... . This is certainly all right. We have also replaced many English terms with more memorable and shorter symbols:

sum: ++
product: **
average: ++:
count: ++1
...

Out of gratitude to the Ocaml developers, we have introduced 2 French words for selection:
avec (with), sans (without)

Where we have found very short memorable known words in a language other than English, we substitute English terms with shorter ones from other languages if those words are known to many people:

true: si (Spanish Italian)
false: no

From the translation of SELECT-FROM-WHERE (gib-aus-mit) are the German words
gib (select) for "give me"
and
aus (from)
have become.

4. **programs are processed from top to bottom and from left to right.**

Programs with loops or general recursion are expressive and powerful, but often difficult to read and understand. Sequential programs are expected to be not so expressive. o++o was also developed to prove the opposite. This requires powerful and expressive operations.

Example: Is 37 not a prime number?
First, all products up to 100 are calculated:

```
Xl:=2 ..50            # 49 Generate numbers
Yl:=2 ..10 at X       # generate 9 numbers for each X value
PRODUCT:=X*Y         # calculate all products
avec PRODUCT <= 100   # select the desired products
gib PRODUCTm         # Sort products and eliminate duplicates
ANSWER:= 37 in PRODUCTm # Is 37 a product?
```

is the comment character.

Readability of programs and tabments is an important problem.
o++o takes this into account as follows:

1. programs can often be written short.
   With the above program for the determination of all products some concepts of o++o can be explained well. If one only wants to know whether 37 is a prime number, this can be formulated much shorter:

```
DIVIDERl:= 2 .. 19
gib ANSWER ANSWER:=37 rest DIVIDER = 0 ! ||
```

The program is certainly easy to read, if you have internalized that || is the existential aggregation and you can use it in the same way as the ++ aggregation.

2. Numbers can also be displayed in Swiss style (e.g.: 12'345'678)
3. lines indented by more than 4 spaces logically belong to the previous line. E.G.:

```
my_marks.tab
gib AVG,(SUBJECT,AVG m)
    AVG:=MARK! ++:   # this line belongs from the logical
                     # point of view still to the previous
rnd 1
```

4. a structured table with the scheme

```
DEPARTMENT, CHIEF, (NAME, SALARY l) m
```

contains each department and boss only once. This not only reduces redundancy, but also improves readability compared to flat tables of this type.

In the chapters it is shown how general and simple the query possibilities of o++o are. Chapter 1 introduces some basic functions of our "pocket calculator". All examples there do not require any stored tables or documents. This does not mean that our o++o app cannot work with files. Tables and documents can also be stored on the smartphone.

First of all, the user must understand what a schema is and what are the tables or documents that belong to this schema. Then it will not be too difficult to grasp the query examples for selection, calculation and restructuring of the first chapters. All operations allow a compact and readable formulation of (complex) queries. They apply to nested lists or sets, and they are new to the database world. Calculations can often be understood as hierarchical "map" functions. Restructuring with the gib clause is very expressive, as it is combined with sort (m, b), duplicate elimination (m), aggregation (++, min, max, ++1, ++:, ||, &&, **,variance).

We know of no other restructuring operation in a commercial product that allows to transform a given hierarchy only by specifying a schema or TT (Tabment Type) of the desired structure. Although the operations in the examples are only applied sequentially, they cover a wide range of applications.

Section 10.4 introduces a "natural" join operation and its un-nested and nested uses. It becomes clear that we do not need the Cartesian product and even the ordinary flat relational join. A simplified notion of recursion is introduced in Chapter 3. With this end-user recursion, appropriate queries can be realized with minimal learning effort. After showing in Chapter 4 that printing two words is not just a syntactic issue, Chapter 6 tries to make clear that o++o is useful for all subjects in school, but especially for mathematics and computer science. It will be made clear that even 9th or 10th grade students can solve problems that are applications of differential and integral calculus. In addition, it is argued that the ordinary division algorithm could be eliminated from the mathematics curriculum. The first "end-user join" (igib) (Section 10.5) roughly speaking extends the restructuring operation to multiple input tables. It requires neither Cartesian product nor (hidden) join conditions. Then, queries to Wikipedia are introduced in chapter 11. Not only table-oriented queries with reference to infoboxes are considered, but also document-oriented ones.

Chapter 17 contains some queries where the result can be interpreted as an image. Roughly speaking, each result table contains the coordinates of points possibly combined with a color value. It is also shown that it is easier to create structured diagrams based on structured tables.

The most important operations of the data model are described in more detail in chapter 10. Section 10.3 contains the description of the restructuring operation, 10.2 the assignment operation and 10.1 the selection.

Acknowledgements:
I would like to thank the following computer scientists for their valuable contributions to our system o++o and previous systems:

Wolfgang Reichstein for the first one-step implementation of the restructuring operation in C for HSQ files,

Dmitri Shamshurko for the first implementation of the first core of the "gib-out-with" construct in a functional style (Caml Light),

Martin Schnabel for the conception and implementation of subroutines and other features,

Andreas Hauptmann for improving many concepts in design and efficiency, especially for query optimization concepts.

Further thanks go to Stephan Schenkl and Mirko Otto for supporting the o++o project.

Thanks to Eicke Redweik for implementing access to a relational DBMS and to Wikipedia and to Jens Winter for implementing a first o++o app for Android.

1 Calculations and spreadsheet applications with o++o

We first present some numerical calculations.

Program 1.1: Addition	Result
`1.23 + 4.56`	`5.79`

Program 1.2: Division	Result
`1:7`	`0.142857142857`

Program 1.3: Division with improved readability	Result
`1:7 '3`	`0.142'857'142'857`

Program 1.4: Division with rounding	Result
`1:7 rnd 3`	`0.143`

Program 1.5: Exponentiation	Result
`3 ^ 20 '3      # or hoch`	`3'486'784'401`

is the comment character. I.e. that "or hoch" does not belong to the program. Comments can be used to explain programs.

Program 1.6: Addition of rational numbers	Result
`3/4 + 1/3`	`13/12`

Program 1.7: Type of the first input value is maintained	Result
`3/4 + 0.3`	`21/20`

Program 1.8: Type of the first input value is maintained	Result
`0.3+3/4`	`1.05`

Program 1.9: Difference or list	Result
`3 - 2 # Note that "3 -2" is a` `      # List of two numbers`	`1`

Program 1.10: Sine of pi : 2	Result
`pi : 2 sin`	`1.`

Program 1.11: Sine of 30 degrees	Result
`30:180*pi sin`	`0.5`

Program 1.12: How many 10-digit binary numbers are there?	Result

`2 ^ 10       # base:2    exponent: 10`	`1024`

Program 1.13: Calculate the edge length of a cube with volume 2	Result
`2 ^ 1/3`	`1.25992104989`

or

Program 1.14: Calculate the edge length of a cube with volume 2 using ordinary division operation	Result
`2 ^(1:3)`	`1.25992104989`

Program 1.15: Sum of 4 numbers	Result
`3.21 4.56 6.88 9.32 ++`	`23.97`

Program 1.16: Sum of numbers from 1 to 100	Result
`1 .. 100 ++`	`5050`

Program 1.17: Product of the numbers from 10 to 40	Result
`10 .. 40 **`	`224844379201911853600532206127677440000000`

You can see from the result that you can process arbitrarily large integers with o++o.

Program 1.18: Maximum of numbers	Result
`1/3 2/7 max`	`1/3`

Program 1.19: Average of several marks	Result
`1 3 2 1 3 4 ++:`	`2.33333333333`

Program 1.20: Introduction of two column names (Output values of two terms simultaneously)	Result
`X:=2 ^ 10 # := : Assignment` `Y:=X : 10`	`X,   Y` `1024 102.4`

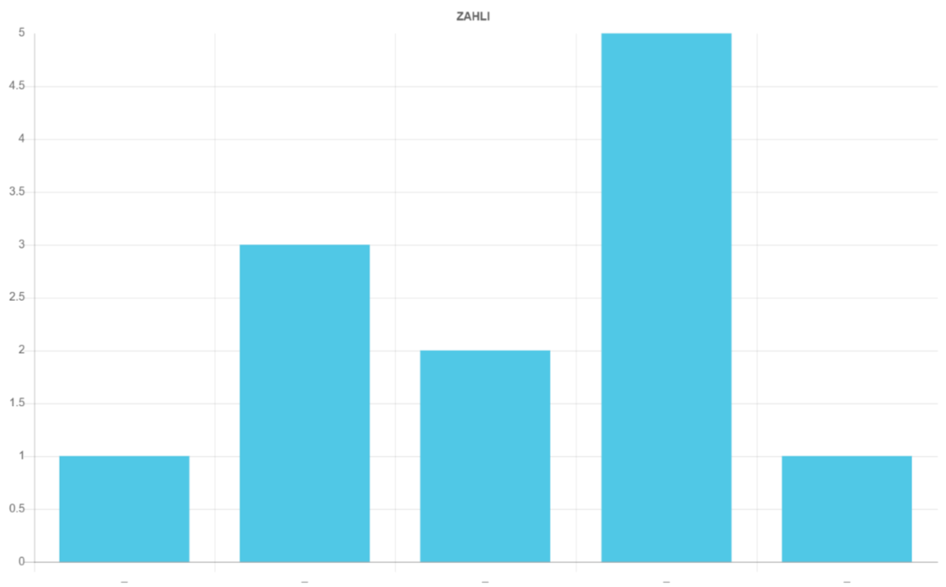
Program 1.21: a pair of two independent terms	Result
`2 sqrt; 3 sqrt # ; separates` `               # stronger than ,`	`PZAHL,        PZAHL` `1.41421356237 1.73205080757`

There are few commas in primary data of tables. This would destroy the readability. Therefore, we do not find commas in .tab files, for example, even if pairs or tuples are represented. However, pairing is represented in the metadata (table headers) of the tables to prevent misunderstandings. PZAHL is a number with a point.

Program 1.22: Comma is an ordinary operation: Calculation from left to right	Result

`2 sqrt,3 sqrt  # the last sqrt`
`               # acts both via`
`               # "2 sqrt" as well`
`               # as over 3`
	`PZAHL,      PZAHL`
	`1.189207115 1.73205080757`

Program 1.23: divrest generates a pair of numbers	Result
`13 divrest 5`	`ZAHL,ZAHL` `2    3`

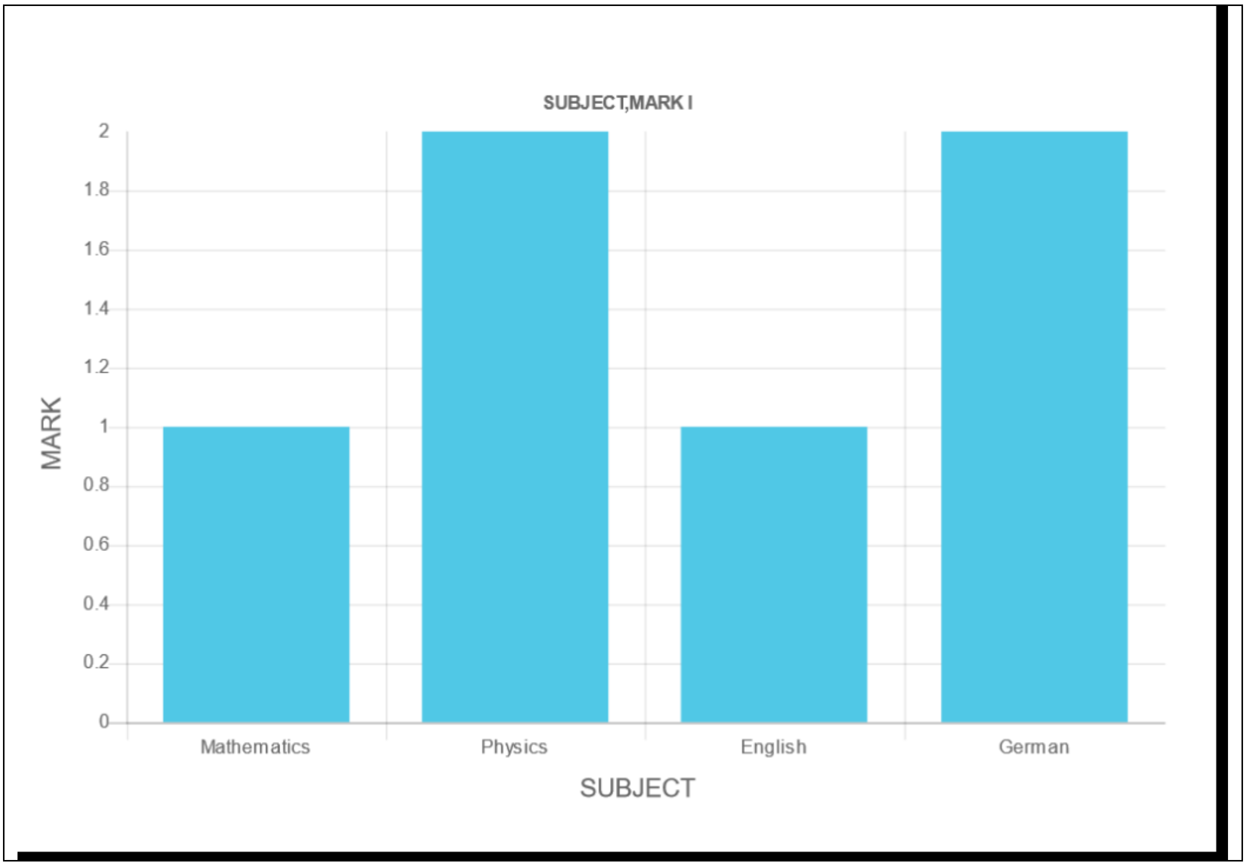
Program 1.24: create a simple diagram with one click
`1 3 2 5 1 # List of numbers`
Result: Diagram (columns) 

SUBJECT,	MARK l
Mathematics	1
Physics	2
English	1
German	2

notes1.tab

The above table represents a list of (SUBJECT,MARK) pairs. It can be created with any text editor or typed into the output field of the o++o interface. l stands for list.

Program 1.25: a simple bar chart with signatures
`notes1.tab`
Result (diagram - Säulen)



It is also possible to enter the following line into the program field of the Otto interface.

```
SUBJECT,MARK l:=Mathematics Physics English German,,1 2 1 2
```

By the operation ,, the both given lists are elementwise connected by comma. The resulting list consists of 4 (WORT,ZAHL) pairs, where the first column is renamed to SUBJECT and the second to MARK.

The basic data of the following query can be generated by the following small structured table. Here l stands for list. It needs the ending tabh, because the marks are arranged horizontally. Lists were invented in Venice (Lista). The single entries (=elements = rows) of the list were arranged one below the other. The subjects are also arranged vertically in noten2.tabh. Simple lists were already arranged horizontally thousands of years ago. A sentence is a list of words, which were essentially arranged horizontally. Since this saves a lot of screen space and paper, simple (single-column) lists in o++o can also be arranged horizontally. This is possible because the list is understood abstractly. This allows o++o to understand JSON lists, for example, even though the list elements are not simply separated by spaces. In questions of the representation of the elements, sets and multisets are equal to lists. However, different parentheses are used.

SUBJECT,	MARKl m
Mathematics	2 1 3
Physics	2 2 3
English	1 4

marks2.tabh

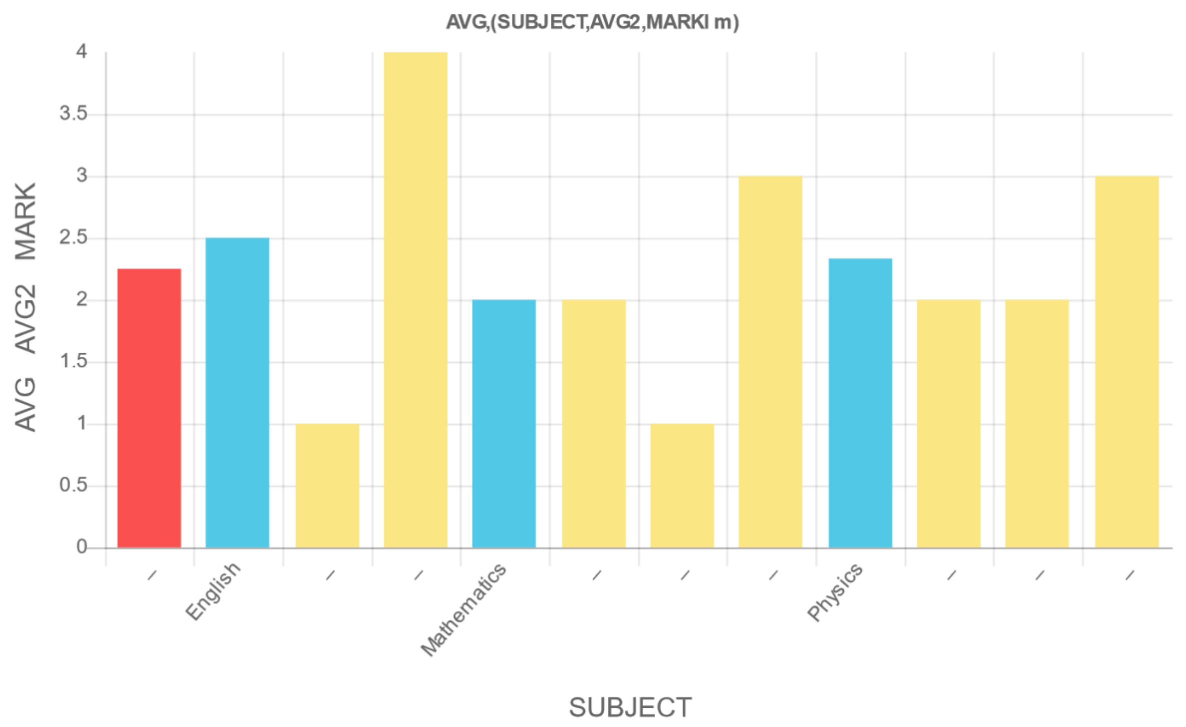
This table can also be generated by the following program line with set brackets { }:

```
SUBJECT,MARKl m:={Mathematics,[2 1 3] Physics,[2 2 3] English,[1 4]}
```

Program 1.26: a structured diagram

```
marks2.tabh
gib AVG,(SUBJECT,AVG,MARKl m)
    AVG:=MARK! ++:
```

Result (diagram columns)



Result (tabh output)

AVG,	(SUBJECT,	AVG2,	MARKl m)
2.25	English	2.5	1 4
	Mathematics	2.	2 1 3
	Physics	2.333333333	2 2 3

The following are examples of a curve discussion using a parabola as an example.

Program 1.27: Calculation of a small table of values of the quadratic function with coefficients 1 -8 13 (x^2 - 8 x +13)

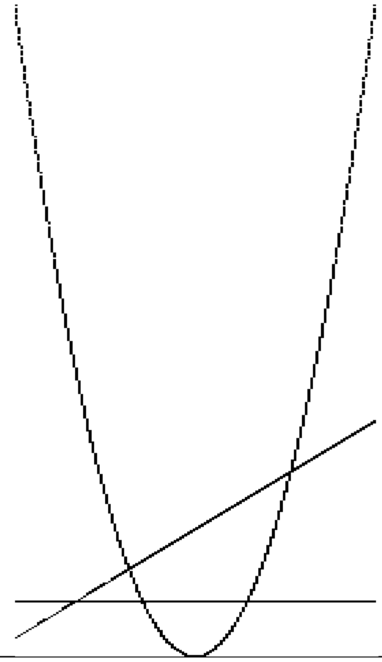
```
Xl:= -2 .. 10
Y := X poly [1 -8 13]
```

Result (tab)

X,	Y l
-2	33
-1	22
0	13
1	6
2	1
3	-2
4	-3
5	-2
6	1
7	6

	8 13 9 22 10 33

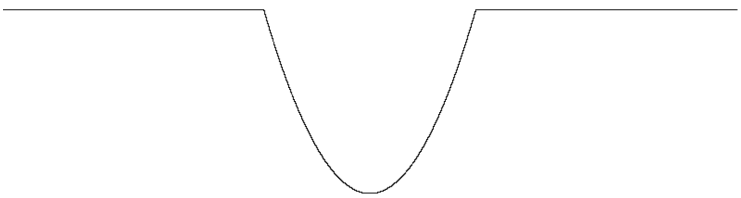
Program 1.28: Expanding the value table so that a function graph can be seen. Draw the graph of the parabola (quadratic function) with the x-axis and the function y=x in the interval [-2 10].	Result (image)

`Xl:= -2 .. 10! 0.01`
`Y := X poly 1 -8 13`
`LINE:= X`
`Y0:= 0*X`
	

Program 1.29: Approximate determination of the (local) minimum of the parabola	Result
`-2 ... 10!0.0001 poly [1 -8 13] min`	-3

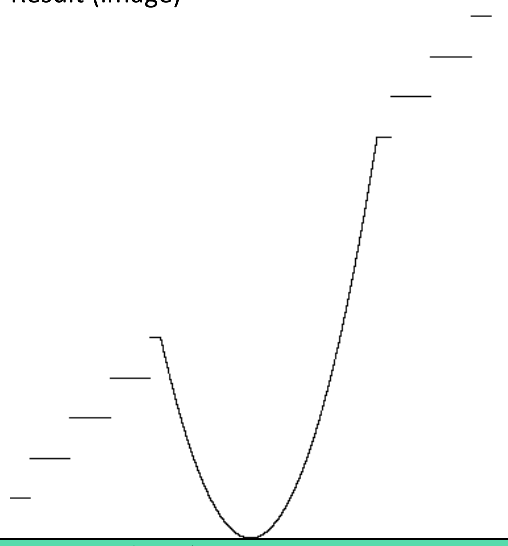
Program 1.30: Approximate determination of the two zeros	Result (tab)

`Xl:= -2 ... 10!0.0001`
`Y := X poly [1 -8 13]`
`avec Y succ * Y <= 0 # succ: successor`
`rnd 7`
	`X,         Y 1`
	`2.2679000  0.0001704`
	`5.7320000 -0.0001760`

Program 1.31: Determining the area under a (composite) function	Result (image) (without 2 last program lines) 
`Xl:= -2 ... 10! 0.0001` `Y := (X poly [1 -8 13],0) min`	Result (tab)

`RECTANGLE:= Y*0.0001` `++ RECTANGLE`	-6.92820323316

If we omit the last two program lines in the following program, the function can be visualized by clicking on bild:

Program 1.32: Determination of the area under a non-continuous function	Result (image) 

`Xl:= -2 ... 10! 0.0001`
`Y := X poly (1 -8 13),(X rnd 0) min`
`RECTANGLE:=Y*0.0001`
`++ RECTANGLE`
	Result (tab)
	21.970089131

Program 1.33: Using the divrest function to output number pairs	Result (tab)
`Xl:=1 ..10` `DIV,REST:=X divrest 3`	X, DIV,REST l 1 0 1 2 0 2 3 1 0 4 1 1 5 1 2 6 2 0 7 2 1 8 2 2 9 3 0 10 3 1

Program 1.34: Determination of all prime numbers up to 70

```
Xl:= 2 .. 35
Yl:= 2 .. 9 at X
PRODUCT:= X*Y
avec PRODUCT <= 70 # avec: with
gib PRODUCTm
PRIMl:= 2 ..70
sans PRIM in PRODUCTm # sans: without
gib PRIMl
```

Result (tabh output):

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
```

Program 1.35: Calculate the circumference of several circles, whose radii are given. The results are to be rounded to 2 digits after the point.

```
4 5 6 2 3.7 9.77 *pi*2 rnd 2
```

Result (tabh)

```
25.13 31.42 37.7 12.57 23.25 61.39
```

You can see that this program can be written in one line.

Program 1.36: Calculating circumference and area of several circles, whose radii are given	Result (tab)
Rl:= 4 5 6 2 3.7 9.77 CIRCUM :=R*pi*2 AREA:=R*R*pi rnd 1	R, CIRCUM, AREA l 4. 25.1 50.3 5. 31.4 78.5 6. 37.7 113.1 2. 12.6 12.6 3.7 23.2 43.0 9.8 61.4 299.9

By Rl:= the name R (called "tag") is assigned to each element of the given list.

An assignment (":=") adds a new column to the specified table. In the above program, the columns CIRCUM and AREA are added one after the other, resulting in a table of type R,CIRCUM,AREA l. l stands for list. Unfortunately, this can easily be confused with the one.

Program 1.37: Calculating perimeter and area of multiple rectangles	Result (tab)
<TAB! A, B l 1.23 5.67 7.65 4.32 9.87 6.54 !TAB> CIRCUM:=A+B*2 AREA:=A*B	A, B, CIRCUM, AREA l 1.23 5.67 13.8 6.9741 7.65 4.32 23.94 33.048 9.87 6.54 32.82 64.5498

The TAB brackets ("<TAB!", "!TAB>") are needed only in the program part of the system. In a file the system recognizes the type by the ending ".tab". In the TAB representation the values must be aligned to the left side of the associated column names.

Program 1.38: Total price of a simple invoice	Result
<TAB! ARTICLE, PRICE l Beer 0.61 Lemonade 0.23 Steak 2.40 !TAB> ++	3.24

Here we simply sum over the numbers in the given table (a list of pairs). The ARTICLE values are words and therefore have no effect on the result. Now we replace ++ with +% 10. This creates a table with 2 columns and three rows (records, tuples). Each number now still contains 10% tip:

Program 1.39: Multiplying a table by a number	Result

<TAB! ARTICLE, PRICE l Beer 0.61 Lemonade 0.23 Steak 2.40 !TAB> +% 10	ARTICLE, PRICE l Beer 0.671 Lemonade 0.253 Steak 2.64

Then you can add again with ++ to get the total (3.564).

Program 1.40: Find the total price of a more complicated calculation using a simple table	Result
<TAB! ARTICLE, PRICE, CNT l Beer 0.61 7 Lemonade 0.23 3 Steak 2.40 4 !TAB> POSPRICE:= PRICE*CNT ++ POSPRICE	14.56

As a result of the assignment, the specified table is extended by a new column with the column name POSPRICE, where each of the three PRICE values is multiplied by the associated CNT value. To determine the total price, a second input value must be passed to the ++ operation. Otherwise, the sum of all nine numbers in the table above would be formed. Both lines of the program can also be replaced by:

`++ PRICE * CNT`

The first input value of an operation, which is at the beginning of a program line, is always the result of the previous program line.

The ":=" character of the assignment is to be distinguished from the equal sign =. For the formulation of conditions the equal sign, as well as <, >, <=, "in" etc. is needed. Conditions are used for selection (filtering of (complex) rows of structured tables).

For example, add a condition
`avec ARTICLE = beer`
or only
`avec beer`
then the final result is the total price for the seven beers. If you want to calculate only the price for the other items instead, use
`sans ARTICLE = beer`
or simply
`sans beer`.

Column names (metadata) must always be written in upper case. The keywords (gib, sans, avec, ...) must always be written in lower case. If you write a word of the primary data always with upper and lower case letters, the program becomes easier to read.
The reference to the aggregation (here ++) results from the header line of the desired table. TOTAL is an aggregation per NAME. Sets (m, m-) are always sorted by the column names specified first.

Program 1.41: Find a weighted average for 3 students and the overall average	Result (tabh)
<TABH!	TOTAL,(NAME, TOT, EXAM1,MARK1 l)

NAME, EXAMl,MARKl l Ernst 1 2 1 2 3 1 3 1 1 Clara 1 1 3 Sophia 1 3 1 !TABH> TOT:=EXAMl ++: *0.6 +(MARKl ++: *0.4) TOTAL:=TOTl ++: rnd 2	1.66 Ernst 1.59 1 2 1 2 3 1 3 1 1 Clara 1.8 1 1 3 Sophia 1.6 1 3 1

Program 1.42: A woman weighs 40 kg plus half her weight. How much does she weigh?	Result
WEIGHTl:= 40 .. 100 avec WEIGHT:2+40=WEIGHT	80

Program 1.43: A bottle with a cork costs one euro and ten cents. The bottle is one euro more expensive than the cork. How much does the bottle cost?	Result
BOTTLEl := 0 .. 110 avec 110-BOTTLE=BOTTLE- 100 # = CORK	BOTTLE 105

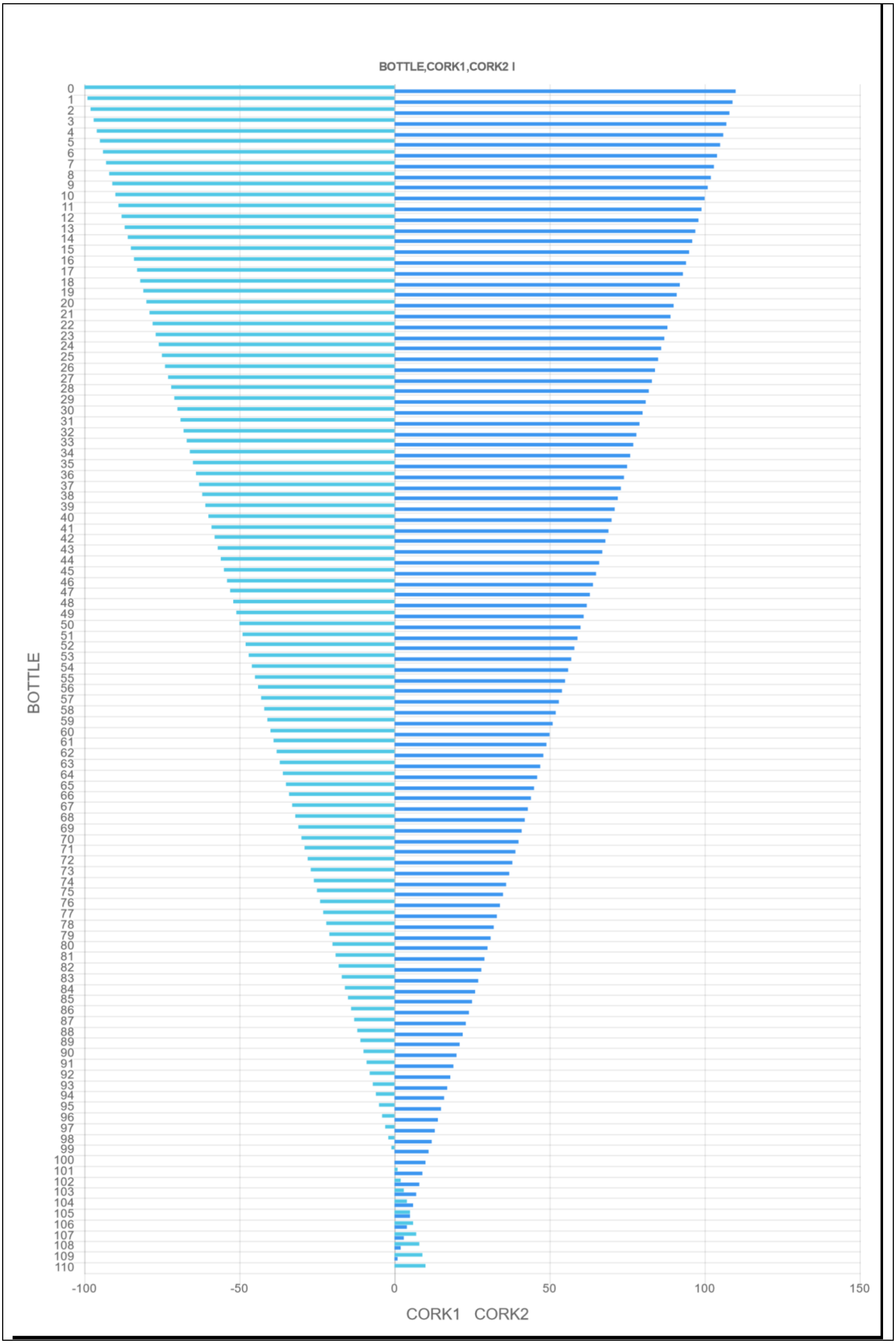
Program 1.44: A bottle with a cork costs one euro and ten cents. The bottle is one euro more expensive than the cork. How much does the cork cost?	Result
BOTTLEl:= 0 .. 110 CORK:= BOTTLE - 100 avec CORK+BOTTLE = 110	BOTTLE, CORK l 105 5

The first assignment gives each of the numbers from 0 to 110 the tag BOTTLE. This is best seen by looking at the ment representation:	If we had written the assignment BOTTLE:= 0 ..110, the BOTTLE tag would appear only once:
<TABM> <BOTTLE>0</BOTTLE> <BOTTLE>1</BOTTLE> <BOTTLE>2</BOTTLE> <BOTTLE>3</BOTTLE> ... <BOTTLE>108</BOTTLE> <BOTTLE>109</BOTTLE> <BOTTLE>110</BOTTLE> </TABM>	<BOTTLE> 0 1 2 3 ... 108 109 110 </BOTTLE>

Program 1.45: A bottle with a cork costs one euro and ten cents. The bottle is one euro more expensive than the cork. How much does the bottle cost?	Result
BOTTLE:= 0 ..110 CORK1 := BOTTLE - 100 CORK2 := 110 - BOTTLE avec CORK1=CORK2	BOTTLE, CORK1, CORK2 l 105 5 5

This solution is advantageous from a methodical point of view, because the first 3 program lines can be displayed by clicking on the image button. You can see that there are 2 straight lines whose intersection is determined by the conditions. You can also click diagram/Balken to get the following result, where it is visible that both bars are equal at 105. Here we had to add the program line:

BOTTLE::=BOTTLE wort



BOTTLE,CORK1,CORK2 I
BOTTLE
CORK1 CORK2
-100
-50
0
50
100
150

2 A savings bank account

The following requests refer to data records of the savings bank. Here the customer can download his data as a csv file. csv files have a very simple structure. Since they contain a lot of quotation marks, they are relatively difficult to read. The otto user does not need to familiarize himself with this syntax. He can view them or parts of the file in the usual way as tab, hsq, ment, web or json files. We consider a file `turnover.csv,` which contains transactions from 3 years.

Program 2.1: How many turnovers are there?	Result
`turnover.csv` `++1`	`162`

Program 2.2: Give the first columns of the first and last transaction!

```
turnover.csv
avec AMOUNT pos=1 | AMOUNT pos- =1
```

Result (tab output):

ORDERACCOUNT,	POSTINGDATE,	VALUEDATE,	POSTINGTEXT,	USAGE....
DE598105327206411	20.07.22	20.07.22	ONLINE REFERRAL	ReNr2
DE598105327206411	13.05.20	13.05.20	ONLINE REFERRAL	Wage

pos determines the position of a tuple. pos starts counting from the beginning with 1 and pos- from the end. "|" is the logical or sign.

Program 2.3: How much money was transferred to the account?	Result

`turnover.csv`
`avec AMOUNT > 0`
`++ AMOUNT`
`'3`
	`110'729.17`

Program 2.4: How much money was transferred from the account?	Result

`turnover.csv`
`avec AMOUNT < 0`
`++ AMOUNT`
`'3`
	`-94'713.65`

Program 2.5: Contrast total revenues and expenditures	Result (tab)

`turnover.csv`
`gib INCOME,EXPENSES`
`    INCOME:= AMOUNT if AMOUNT > 0!++`
`    EXPENSES:= AMOUNT if AMOUNT< 0!++`
`'3`
	`INCOME ,    EXPENSES`
	`110'729.17  -94'713.65`

Program 2.6: How much was transferred to Ms. Heyer in total?	Result
`turnover.csv` `avec Heyer`	`-54'538.28`

++ AMOUNT '3	

Here the user must know his data. If there are two Heyer's, the above result is certainly not the desired one. One could then add the first name:

avec Heyer Erika

or

avec Heyer & Erika

You can also use the account number, but then the program is not so well readable, because most people cannot remember an account number or IBAN.

Program 2.7: How much was transferred to Ms. Heyer in each year and month?

```
turnover.csv
avec Heyer
D,MONTH,YEAR:=VALUTADATUM zahltrip
gib YEAR,SUM,(MONTH,SUM m) m
    SUM:=AMOUNT ! ++
'3

rnd 2
```

Result (tab)

YEAR,	SUM ,	(MONTH,	SUM2 m)m
20	-16'807.22	5	-1'967.66
		6	-1'777.31
		7	-1'943.89
		8	-2'110.27
		9	-1'833.69
		10	-3'189.39
		11	-1'973.48
		12	-2'011.53
21	-20'727.13	1	-2'011.53
		2	-1'911.90
		3	-2'006.17
		4	-2'443.65
		5	-2'438.02
		6	-583.32
		8	-812.47
		9	-2'636.73
		10	-993.13
		11	-2'635.68
		12	-2'254.53
22	-17'003.93	1	-1'630.30
		2	-2'202.03
		3	-1'324.38
		4	-1'438.72
		5	-4'460.36
		6	-2'999.59
		7	-2'948.55

Program 2.8: Give a comparison of the income and expenses for each year!

```
turnover.csv
YEAR:=20 wort + (VALUTADATUM subtext 7!2)
gib YEAR,PLUS,MINU,SUM m
    PLUS:=AMOUNT if AMOUNT>0!++
    MINU:=AMOUNT if AMOUNT<0!++
    SUM:=AMOUNT!++
'3
```

Result (tab output):

```
YEAR, PLUS ,    MINU ,     SUM m
2020  24'921.   -23'257.11  1'663.89
2021  50'468.04 -34'274.22 16'193.82
2022  35'340.13 -37'182.32 -1'842.19
```

subtext needs 3 input values. Here VALUTADATUM is the first, the initial character number 7 the second and the length of the desired string 2 the third. VALUTADATUM is here a word constructed in German date notation, e.g.: 18.06.21. In the above example, however, the year is to be output with 4 digits. For this, the word "20" must be concatenated with the two digits that subtext determines.

Program 2.9: Give me for each month of the year 2021 the income and expenses with the larger transfers!

```
turnover.csv
avec VALUTADATUM subtext 7!2 = 21
MONTH:=VALUTADATUM nthzahl 2
USE:=VERWENDUNGSZWECK subtext 3!6
RECIPIENT:=BEGUENSTIGTER_ZAHLUNGSPFLICHTIGER subtext 3!6
gib MONTH,PLUS,MINU,(AMOUNT,USE,RECIPIENT b-) m
    PLUS:=AMOUNT if AMOUNT>0 ! ++
    MINU:=AMOUNT if AMOUNT<0 ! ++
'3
avec AMOUNT! AMOUNT abs>2'000
rnd 2
```

Result (hsq output):

```
MONTH,PLUS,MINU,(BETRAG,USE,RECIPIENT b-) m
MONTH PLUS MINU
 AMOUNT USE RECIPIENT
01 0 -3'536.75
02 18'391.40 -1'922.80
 11'119.40 SZAHLU vestit
 7'272.00 SZAHLU vestit
03 0 -2'969.82
04 0 -3'238.44
05 0 -3'572.81
06 3'513.33 -2'417.52
 3'513.33 /2018/ vestit
07 279.37 -2'001.98
08 410.68 -2'258.82
09 0 -3'932.65
10 27'527.11 -1'152.42
 17'413.36 971240 vestit
 10'113.75 971240 vestit
11 0 -2'764.63
12 346.15 -4'505.58
```

For reasons of presentability, corresponding data of the recipient and the purpose of use have been shortened.

If the last line were replaced by

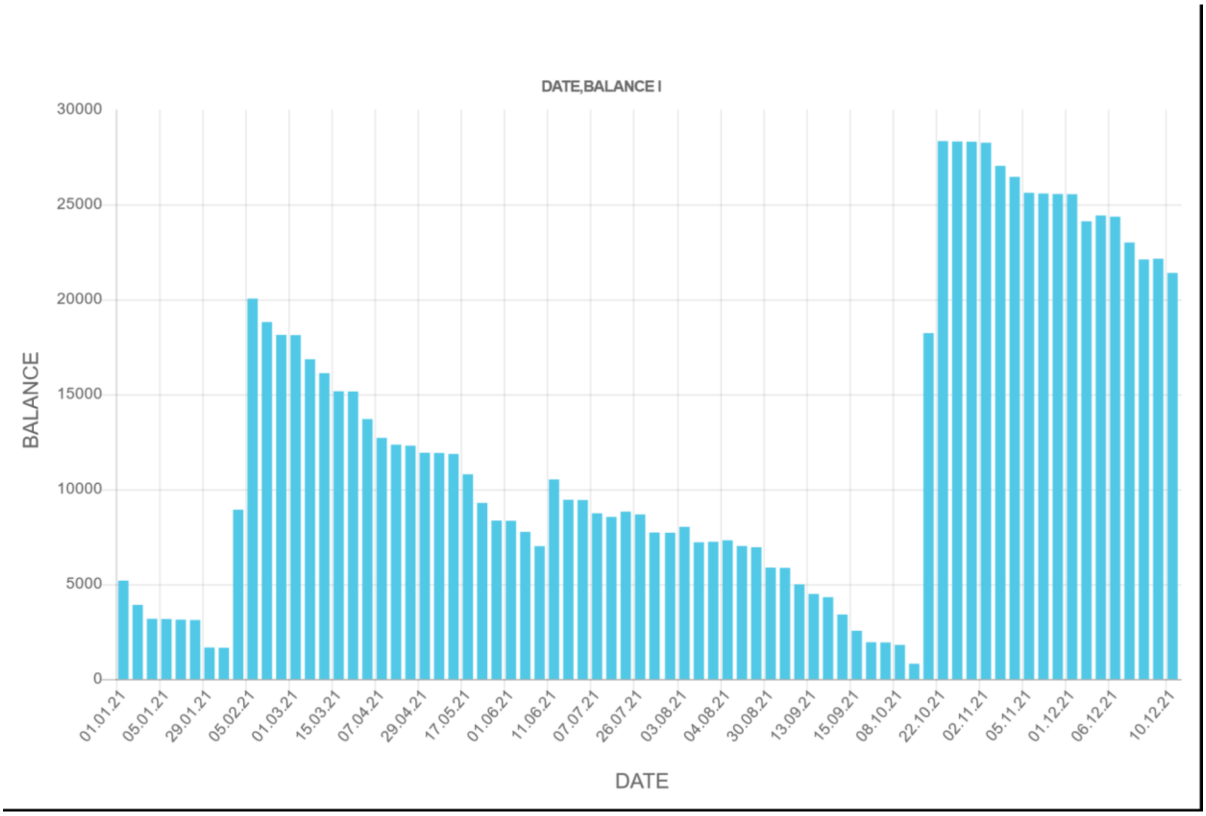
```
avec AMOUNT abs > 2'000
```

replace, only the data for months 2, 6 and 10 would remain, since the others do not contain any major transactions (over 2,000 euros).

Program 2.10: Output the account balances of 2021 as a bar chart output!

```
turnover.csv
rename VALUTADATUM!DATE
avec DATE subtext 7!2=21
gib DATE,BETRAG l-
BALANCE:=  5'200 +BETRAG for BALANCE pred +BETRAG at BETRAG
gib DATE,BALANCE l
```

Result (bar chart):



Here it is assumed that the initial account balance is 5'200. This number is simply added to the first AMOUNT (BETRAG) value in the above example.

The query possibilities of an account file and other files depend on the existing data. If, for example, no name for the recipient is given in the purpose of use of the data records, it is not possible to write well readable queries. The better the data material, the simpler the o++o programs become and the more queries are possible. But this also makes clear that the one who knows the input data can write the best o++o programs. A computer scientist, who wants to program general evaluations of such data, will never be able to make the variety possible, which an end user reaches, who knows the contents of the records exactly. The intended use alone offers many possibilities to improve the evaluations, which are certainly not yet exhausted by many.

The importance of a simple query language will be magnified when money transactions in Germany are also completely cashless. If everyone has access to the data on their purchases at a supermarket or gas station, they will be able to determine exactly when and on what they spent their money.

3 Table Recursion - Exponential Growth

Recursion is a powerful tool to describe functions or data structures in a short form. It is especially used in functional languages like OCaml or HASKEL. We introduce a type of "forward recursion" that is easy to use. An initial value is always described by a value or a term and the following values result from the direct predecessor by means of a second term, respectively. All generated values are visible in the result table.

In this way one can describe exponential growth. EXP9 and EXP1 are the program lines for this. At the same time, these two columns in the tab representation contain the growth values. One percent growth is exponential growth if compound interest is taken into account. This is given in both formulas. If one adds in each case only 9% or 1% of 100 to the predecessor, then the interest of the interest is not considered. This would be the growth if one takes the interest from the interest every year. Our formulas for LIN9 and LIN1 correspond to this. These straight-line formulas and the exponential function curves differ here only at one point. If the operation + is replaced by +%, linear growth becomes exponential.

As is well known, exponential growth is far superior to any other growth and thus especially to linear growth. The fact that nine percent interest yields a far better total amount after 20 years than one percent is shown by the last line of the table (€560 versus €122). Without compound interest, the results are €280 and €120, respectively. To better compare with polynomial growth, we have included a parabola.

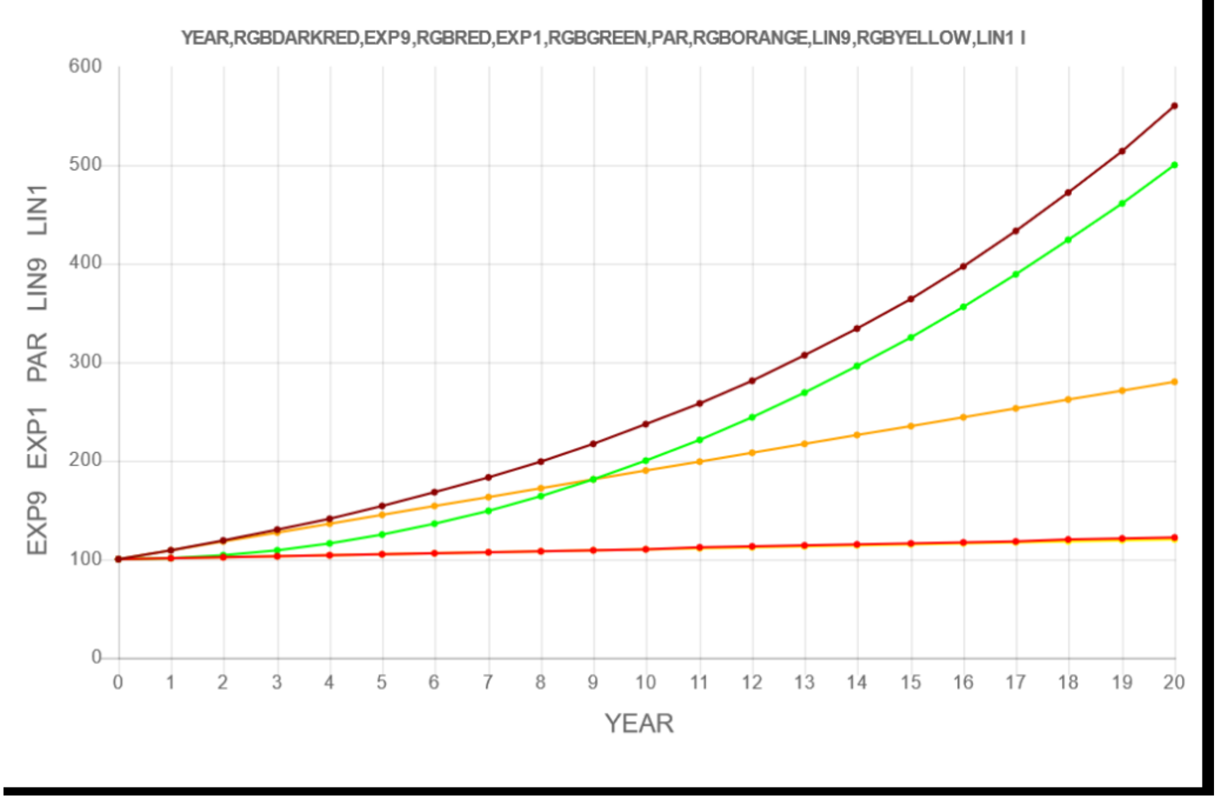
The green parabola obviously shows a similar behavior in this range of 20 years as the exponential growth of 9 percent (dark red). In the next example we will see that this changes completely if we look at 200 years instead of 20. The yellow curve (1 % without compound interest) and the red curve (exponential growth 1 %) practically did not differ at all.

It should already be mentioned here that the curves are "distorted" so that they look nicer. In school, LIN1 would have to be drawn with an angle of 45°. LIN9 would be almost vertical with an angle of more than 83°. If you did that, the values of the fast-growing functions would have no place on the paper or screen, or you would have to shorten the x-axis (here YEAR) accordingly. But then it would look as if all points and curves were vertical. This undistorted real representation of the points is realized by the output 'bild'. This doesn't look nice, but people should be confronted with reality from time to time. Then they can also better classify the visualizations below.

Program 3.1: How does an amount of 100 Euro develop with a "simple" and normal interest rate of 1% and 9%? and with quadratic growth within 20 years.

```
YEAR1:= 0 .. 20
EXP9 :=  100. for EXP9 pred +% 9 at YEAR
EXP1 :=  100. for EXP1 pred +% 1 at EXP9
PAR  := YEAR * YEAR + 100
LIN9 :=  100. for LIN9 pred + 9 at PAR
LIN1 :=  100. for LIN1 pred + 1 at LIN9
rnd 0
YEAR::=YEAR wort
RGBDARKRED :=darkred leftat EXP9
RGBRED     :=red     leftat EXP1
RGBGREEN   :=green   leftat PAR
RGBORANGE  :=orange  leftat LIN9
RGBYELLOW  :=yellow  leftat LIN1
```

Result (line graph):



Result (tab output):

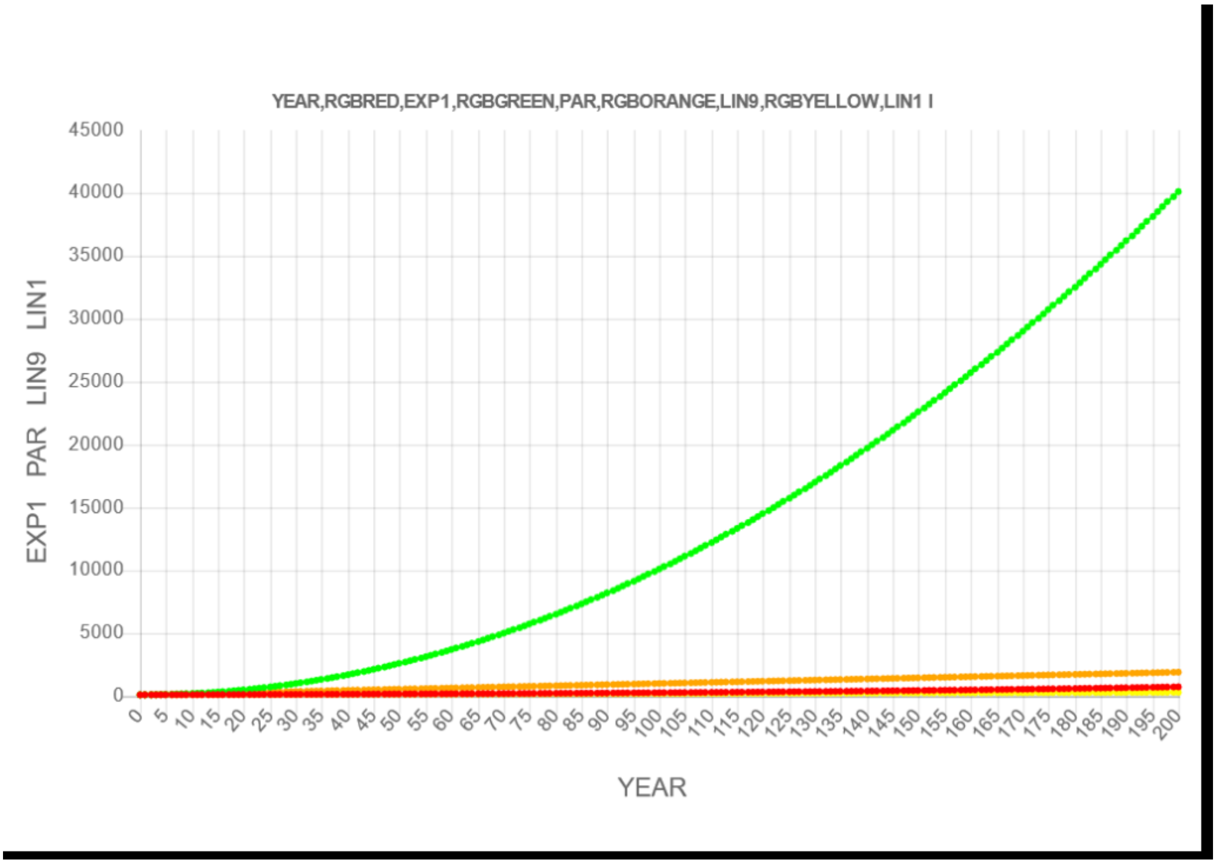
```
YEAR,EXP9,EXP1,PAR,LIN9,LIN1 l
   0 100. 100. 100 100. 100.
   1 109. 101. 101 109. 101.
   2 119. 102. 104 118. 102.
   3 130. 103. 109 127. 103.
   4 141. 104. 116 136. 104.
   5 154. 105. 125 145. 105.
   6 168. 106. 136 154. 106.
   7 183. 107. 149 163. 107.
   8 199. 108. 164 172. 108.
   9 217. 109. 181 181. 109.
  10 237. 110. 200 190. 110.
  11 258. 112. 221 199. 111.
  12 281. 113. 244 208. 112.
  13 307. 114. 269 217. 113.
  14 334. 115. 296 226. 114.
  15 364. 116. 325 235. 115.
  16 397. 117. 356 244. 116.
  17 433. 118. 389 253. 117.
  18 472. 120. 424 262. 118.
  19 514. 121. 461 271. 119.
  20 560. 122. 500 280. 120.
```

The new column EXP9 is defined by two formulas. The first element of the list of years is assigned the value of the first formula. The second value is calculated by the second formula, where "EXP9 pred" is the value of the predecessor. Therefore, we get 100 +% 9=109 for the second value. The third value is again calculated by the second formula, but now we have to calculate 109 +% 9=118.81 (rounded to 119). In the same way, all the following values are calculated value by value using the second formula. The rounding does not cause any inaccuracies, because it is done after all calculations.

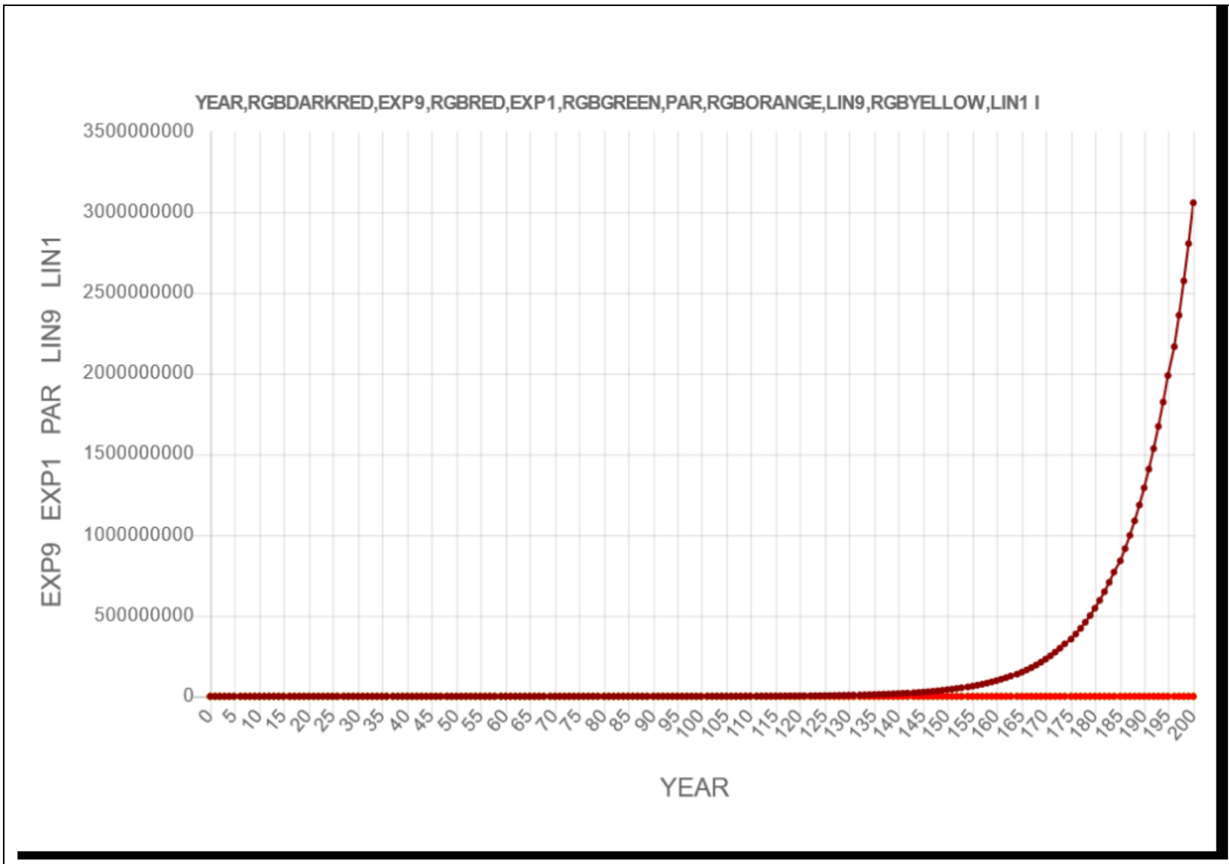
Program 3.2: How does an amount of 100 Euro develop with a "simple" and normal interest rate of 1 % and 9 % and with quadratic growth within 200 years.

```
YEARl := 0 .. 200
#EXP9  :=  100. for EXP9 pred +% 9 at YEAR
EXP1  :=  100. for EXP1 pred +% 1 at YEAR #EXP9
PAR   := YEAR * YEAR + 100
LIN9  :=  100. for LIN9 pred + 9 at PAR
LIN1  :=  100. for LIN1 pred + 1 at LIN9
rnd 0
#avec YEAR rest 10 = 0 this condition was applied to reduce the volume of
#tab output.
YEAR::=YEAR wort
'3
#RGBDARKRED :=darkred leftat EXP9
RGBRED     :=red    leftat EXP1
RGBGREEN   :=green  leftat PAR
RGBORANGE :=orange leftat LIN9
RGBYELLOW :=yellow leftat LIN1
```

Result (line chart without EXP9)



Result (line chart with EXPO9)



Result (tab output):

YEAR	,EXP9	,EXP1	,PAR	,LIN9	,LIN1 l
0	100.	100.	100	100.	100.
10	237.	110.	200	190.	110.
20	560.	122.	500	280.	120.
30	1'327.	135.	1'000	370.	130.
40	3'141.	149.	1'700	460.	140.
50	7'436.	164.	2'600	550.	150.
60	17'603.	182.	3'700	640.	160.
70	41'673.	201.	5'000	730.	170.
80	98'655.	222.	6'500	820.	180.
90	233'553.	245.	8'200	910.	190.
100	552'904.	270.	10'100	1'000.	200.
110	1'308'925.	299.	12'200	1'090.	210.
120	3'098'702.	330.	14'500	1'180.	220.
130	7'335'754.	365.	17'000	1'270.	230.
140	17'366'396.	403.	19'700	1'360.	240.
150	41'112'576.	445.	22'600	1'450.	250.
160	97'328'419.	491.	25'700	1'540.	260.
170	230'411'765.	543.	29'000	1'630.	270.
180	545'468'442.	600.	32'500	1'720.	280.
190	1'291'322'174.	662.	36'200	1'810.	290.
200	3'057'029'208.	732.	40'100	1'900.	300.

The green parabola is not visible in the second image. The points are behind the other non-dark red points. Therefore, the parabola looks like a straight line here. The straight line turns into a fast-growing curve when the even faster growing dark red exponential curve is taken out of the picture. This can only be understood by comparing the scalings of the ordinates (Y-axes).

Program 3.3: The chess board problem: Place a grain of wheat on the first square, two on the second, 4 on the third, then eight, and so on. This exponential growth is compared with the polynomial X^8 .

```
Xl    := 1 .. 64
FIELD :=  1 for FIELD pred *2 at X
HIGH8 := X ^ 8
FIELD::= FIELD if X<30 ! (FIELD div 1'000'000)
HIGH8::= HIGH8 if X<30 ! (HIGH8 div 1'000'000)
'3
```

Result (tab):

```
X  ,FIELD                  ,HIGH8           l
 1                    1                   1
 2                    2                 256
 3                    4               6'561
 4                    8              65'536
 5                   16             390'625
 6                   32           1'679'616
 7                   64           5'764'801
 8                  128          16'777'216
 9                  256          43'046'721
10                  512         100'000'000
11                1'024         214'358'881
12                2'048         429'981'696
13                4'096         815'730'721
14                8'192       1'475'789'056
15               16'384       2'562'890'625
16               32'768       4'294'967'296
17               65'536       6'975'757'441
18              131'072      11'019'960'576
19              262'144      16'983'563'041
20              524'288      25'600'000'000
21            1'048'576      37'822'859'361
22            2'097'152      54'875'873'536
23            4'194'304      78'310'985'281
24            8'388'608     110'075'314'176
25           16'777'216     152'587'890'625
26           33'554'432     208'827'064'576
27           67'108'864     282'429'536'481
28          134'217'728     377'801'998'336
29          268'435'456     500'246'412'961
30                  536             656'100
31                1'073             852'891
32                2'147           1'099'511
33                4'294           1'406'408
34                8'589           1'785'793
35               17'179           2'251'875
36               34'359           2'821'109
```

```
37             68'719       3'512'479
38            137'438       4'347'792
39            274'877       5'352'009
40            549'755       6'553'600
41          1'099'511       7'984'925
42          2'199'023       9'682'651
43          4'398'046      11'688'200
44          8'796'093      14'048'223
45         17'592'186      16'815'125
46         35'184'372      20'047'612
47         70'368'744      23'811'286
48        140'737'488      28'179'280
49        281'474'976      33'232'930
50        562'949'953      39'062'500
51      1'125'899'906      45'767'944
52      2'251'799'813      53'459'728
53      4'503'599'627      62'259'690
54      9'007'199'254      72'301'961
55     18'014'398'509      83'733'937
56     36'028'797'018      96'717'311
57     72'057'594'037     111'429'157
58    144'115'188'075     128'063'081
59    288'230'376'151     146'830'437
60    576'460'752'303     167'961'600
61  1'152'921'504'606     191'707'312
62  2'305'843'009'213     218'340'105
63  4'611'686'018'427     248'155'780
64  9'223'372'036'854     281'474'976
```

You can see that the polynomial on the sixth field has already exceeded the million, but the exponential function is only at 32. In the last line, on the other hand, it becomes clear that the exponential function is larger than the polynomial value by a factor of about 10'000. From position 30 we omitted the last 6 digits to improve the comparability of such large numbers.

Program 3.4: Calculate the total growth of the gross domestic product in West Germany, East Germany, and China in the years from 1992 to 2014 using the growth data given.

```
<TAB!
YEAR, WGER,  EGER, CHIN l
1988   0.      0.    0.
1989   3.9     1.85  4.2
1991  11.09  -47.8  13.56
1992   1.7     6.2  14.3
1993  -2.6     8.7  13.9
1994   1.4     8.1  13.1
1995   1.4     3.5  11.
1996   0.6     1.6   9.9
1997   1.5     0.5   9.2
1998   2.3     0.2   7.8
1999   2.1     1.8   7.6
2000   3.1     1.2   8.4
2001   1.1    -0.6   8.3
2002   0.1     0.2   9.1
2003  -0.1    -0.3  10.
```

```
2004   1.6     1.3  10.1
2005   0.8    -0.2  11.3
2006   3.8     3.4  12.7
2007   3.3     2.9  14.2
2008   1.      0.6   9.6
2009  -6.1    -3.9   9.2
2010   4.3     3.2  10.6
2011   3.8     1.9   9.5
2012   0.4     0.6   7.7
2013   0.1    -0.1   7.7
2014   1.6     1.4   7.4
!TAB>
avec YEAR>1991
WGERM:=  100. for WGERM pred   +% WGER at CHIN
EGERM:=  100. for EGERM pred   +% EGER at WGER
CHINA:=  100. for CHINA pred +% CHIN   at EGERM
rnd 1
avec YEAR>1991
gib YEAR,EGERM,WGERM,CHINA l
```

Result (tab output):

```
YEAR ,EGERM ,WGERM ,CHINA  l
1992 100.0  100.0  100.0
1993 108.7   97.4  113.9
1994 117.5   98.8  128.8
1995 121.6  100.1  143.0
1996 123.6  100.7  157.1
1997 124.2  102.3  171.6
1998 124.4  104.6  185.0
1999 126.7  106.8  199.0
2000 128.2  110.1  215.8
2001 127.4  111.3  233.7
2002 127.7  111.4  254.9
2003 127.3  111.3  280.4
2004 128.9  113.1  308.8
2005 128.7  114.0  343.7
2006 133.1  118.3  387.3
2007 136.9  122.3  442.3
2008 137.7  123.5  484.8
2009 132.4  115.9  529.3
2010 136.6  120.9  585.5
2011 139.2  125.5  641.1
2012 140.0  126.0  690.4
2013 139.9  126.2  743.6
2014 141.9  128.2  798.6
```

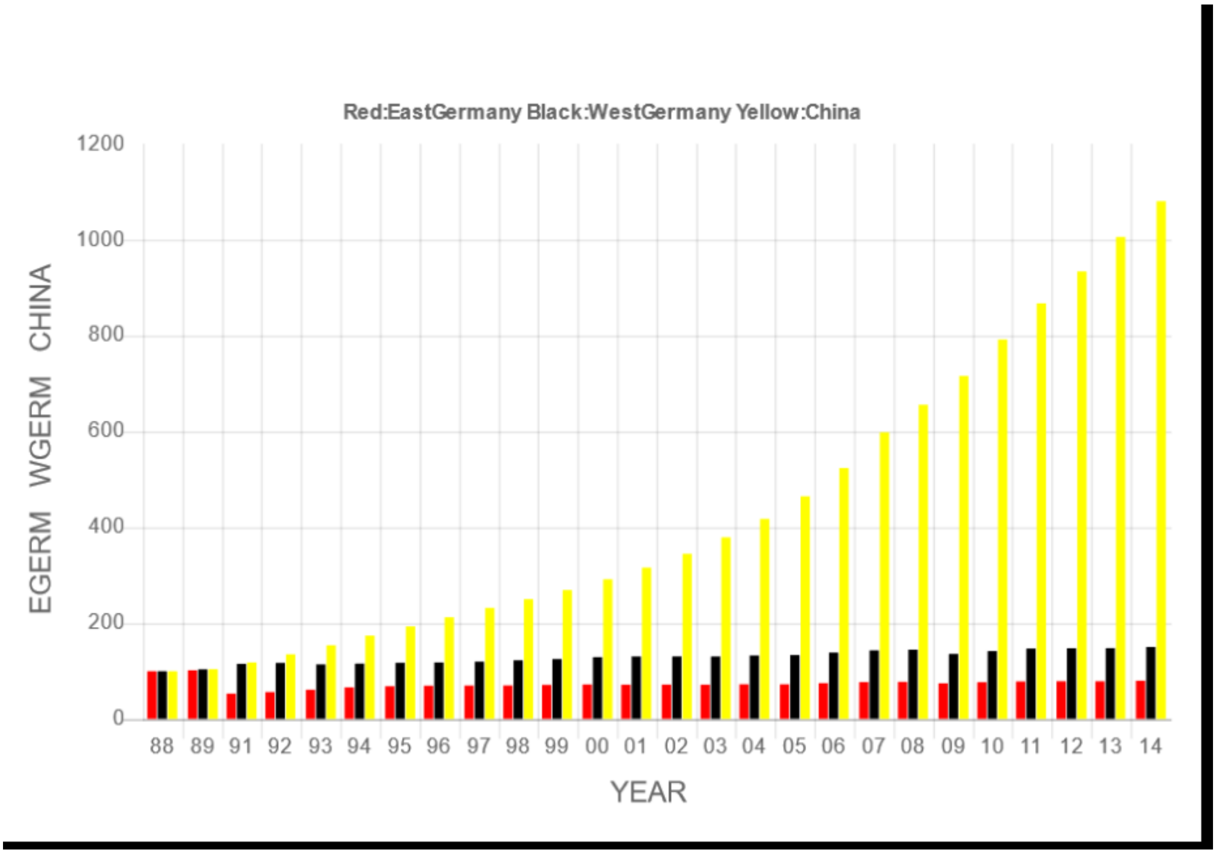
With a total growth of 100 to 142, East Germany is clearly better in this time interval than West Germany with a growth of 100 to 128. Now let the condition YEAR>1991 be dropped. Furthermore, we assume that the above data enclosed in HSQ brackets are in the file growth.tab.

Program 3.4: Calculate the growth of the gross domestic product in East Germany, West Germany and China in the years 1988 to 2014 with the indicated growth.

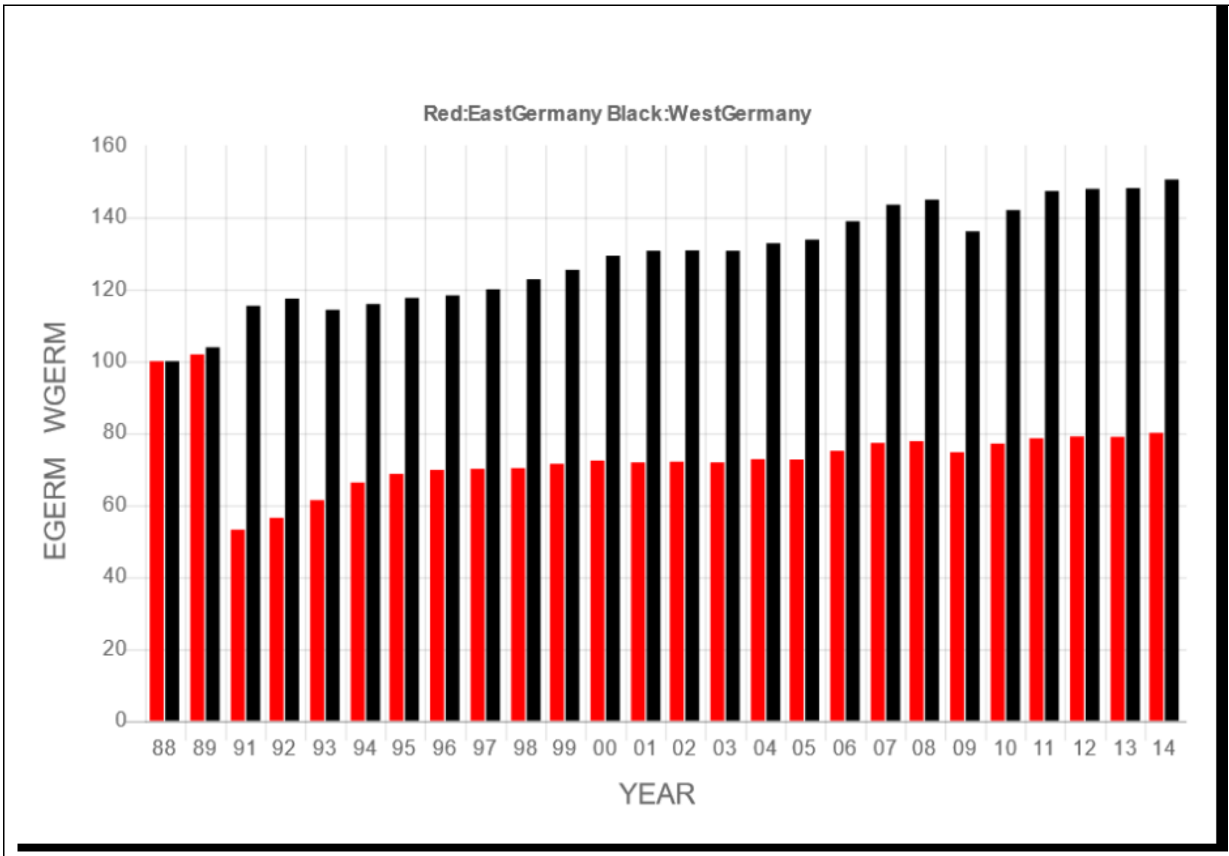
```
growth.tab
TITLE :="Red:EastGermany Black:WestGermany Yellow:China"
WGERM :=  100. for WGERM pred +% WGER at CHIN
```

```
EGERM :=  100. for ODE pred +% EGER at WGERM
CHINA :=  100. for CHINA pred +% CWA at EGERM
rnd 1
YEAR::= YEAR wort subtext 3!2
gib TITLE,(YEAR,EGERM,WGERM,CHINA l)
RGB:=red    leftat EGERM
RGB:=black  leftat WGERM
RGB:=yellow leftat CHINA
```

Result (bar chart):



Result excluding China (bar chart):



Result (tab output):

TITEL	,(YEAR	,RGB	,EGERM	,RGB	,WGERM	,RGB	,CHINA 1)
Red:EastG...	88	1.,0.,0.	100.0	0.,0.,0.	100.0	1.,1.,0.	100.0
	89	1.,0.,0.	101.9	0.,0.,0.	103.9	1.,1.,0.	104.2
	91	1.,0.,0.	53.2	0.,0.,0.	115.4	1.,1.,0.	118.3
	92	1.,0.,0.	56.5	0.,0.,0.	117.4	1.,1.,0.	135.3
	93	1.,0.,0.	61.4	0.,0.,0.	114.3	1.,1.,0.	154.1
	94	1.,0.,0.	66.3	0.,0.,0.	115.9	1.,1.,0.	174.2
	95	1.,0.,0.	68.7	0.,0.,0.	117.6	1.,1.,0.	193.4
	96	1.,0.,0.	69.8	0.,0.,0.	118.3	1.,1.,0.	212.5
	97	1.,0.,0.	70.1	0.,0.,0.	120.0	1.,1.,0.	232.1
	98	1.,0.,0.	70.3	0.,0.,0.	122.8	1.,1.,0.	250.2
	99	1.,0.,0.	71.5	0.,0.,0.	125.4	1.,1.,0.	269.2
	00	1.,0.,0.	72.4	0.,0.,0.	129.3	1.,1.,0.	291.8
	01	1.,0.,0.	71.9	0.,0.,0.	130.7	1.,1.,0.	316.1
	02	1.,0.,0.	72.1	0.,0.,0.	130.8	1.,1.,0.	344.8
	03	1.,0.,0.	71.9	0.,0.,0.	130.7	1.,1.,0.	379.3
	04	1.,0.,0.	72.8	0.,0.,0.	132.8	1.,1.,0.	417.6
	05	1.,0.,0.	72.7	0.,0.,0.	133.8	1.,1.,0.	464.8
	06	1.,0.,0.	75.1	0.,0.,0.	138.9	1.,1.,0.	523.8
	07	1.,0.,0.	77.3	0.,0.,0.	143.5	1.,1.,0.	598.2
	08	1.,0.,0.	77.8	0.,0.,0.	144.9	1.,1.,0.	655.6
	09	1.,0.,0.	74.7	0.,0.,0.	136.1	1.,1.,0.	715.9
	10	1.,0.,0.	77.1	0.,0.,0.	142.0	1.,1.,0.	791.8
	11	1.,0.,0.	78.6	0.,0.,0.	147.3	1.,1.,0.	867.1
	12	1.,0.,0.	79.1	0.,0.,0.	147.9	1.,1.,0.	933.8
	13	1.,0.,0.	79.0	0.,0.,0.	148.1	1.,1.,0.	1005.7
	14	1.,0.,0.	80.1	0.,0.,0.	150.5	1.,1.,0.	1080.2

We have hidden China in the second chart so that it is easier to see Germany's two growth data. For example, East Germany produces less than in GDR times. The banking crisis had a major negative impact on the East German economy, even though East Germany does not have a bank, ... Too much information can obscure what seems to be essential.

4 Hello otto - gimmick

Program 4.1: Output two words out.
`Hello otto`
`Result (tabh)`
`WORT1`
`Hello otto`

Program 4.2: Give a pair of two words from.
`Hello, otto`
`Result (tab)`
`WORT, WORT`
`Hello otto`

Program 4.3: Output a text with spaces.
`"Hello Otto"`
`Result (tab)`
`TEXT`
`Hello otto`

Program 4.4: Concatenate two words with spaces.
`Hello + " " + otto`
Result (tab)
TEXT
`Hello otto`

Program 4.5: Give a greeting with a list of two words.
`GREETING := Hello otto`
Result (ment)
TABMENT! GREETING `GREETING! WORT1`

`<GREETING>`
`Hello`
`otto`
`</GRUSS>`

Program 4.6: Output two words each with its own column name.
DEAR:=Hello GREETING:=otto
Result (tab)
`DEAR, GREETING`
`Hello otto`

Program 4.7: Output two words with one column name.
GREETING:= "Hello otto"
Result (tabh)
`GREETING`
`Hello otto`

Program 4.8: Sort a set of words.
GREETING:= {otto Hello}
Result (tabh)

`TABMENT! GREETING`
`GREETING! WORTl`
`GREETING`

`Hello otto`

Program 4.9: Represent one word by metadata and the other by primary data.
HELLO := otto
Result (tab)
`HELLO`
`otto`

5 o++o for kindergarten?

The stroke list is historically the first representation of a number. It could already be a million years old. Notched wood has been shown to be 150 thousand years old. Concepts first developed in history are usually simpler than later concepts. That's why tally charts should have a broader scope even in kindergarten.

The following goals could be pursued with the use of o++o in kindergarten:

1. By presenting decimal numbers and stroke lists at the same time, a child can better appreciate the magnitude of numbers. For example, the number one hundred differs from the number ten only by one digit zero. The corresponding stroke lists, however, differ considerably.
2. The operation symbols + * - : could be taught. They are probably easier to explain on stroke lists. The stroke lists could be converted to decimals and vice versa.
3. The algorithm behind the stroke list operation (gib statement) could be taught using appropriate examples.
4. Can preschool children formulate appropriate o++o programs? Here one should imagine that an app is developed that replaces letters or words with pictures.

5.1 Stroke Lists

Counting animals of different species could give the following small intermediate table:

```
Elephant | | | |
Deer     | |
Pig      | | |
```

If another deer comes, a stroke is added to the second line. If, on the other hand, a turkey comes, a new line must be added with the name turkey and a stroke at the end.

This already poses many problems, although preschoolers can already create such a table if the words have been replaced by pictures or single letters. It is not clear how many columns this table has if only "normal" tables are considered. If we allow structured tables, we can say that this table contains a column ANIMAL and a column STROKE, but the values of the column STROKE can be "repeated" for each animal. An associated schema `ANIMAL,STROKEl m` or

`ANIMAL,STROKEl l` would express this. Where l is an abbreviation for list and m stands for set. These symbols are again used postfix, i.e. they are placed after. The m is necessary in a gib part so that each animal appears only once in the target table.

Since many children are interested in cars, one could count cars analogously to counting animals. This could result in the following table:

```
Golf     | | | |
A6       | | |
Polo     | | | | |
Wartburg | | |
A8       | |
```

Is it possible in kindergarten to increase the structural depth of the table when counting? Then the following (hsqh-) table could have been created:

```
VW | | | | | | | | |
 Golf | | | |
 Polo | | | | |
```

```
Audi | | |
 A6 | | |
 A8 | |
IFA | | |
 Wartburg | | |
```

5.2 The conversion operations zahl and |l

As a dash (stroke) o++o uses the | (or) character. Several such characters must be enclosed in square brackets on input so that they are interpreted by o++o as a list of strokes. We will illustrate the operations in the following text with self-explanatory examples.

Program 5.2.1: Stroke list to number	Result
`[			] zahl`	3

Convert a number into a list of strokes:

Program 5.2.2: Number to tally list	Result
`4 |l`	`|		|`

5.3 The operations + *

Different representations of an addition task. The first input type again determines the output type.

Programs 5.3.1: Four plus four	Results
`4 + 4`	8
`4 + [				]`	8
`[				] + [				]`	`|						|`
`[				] + 4`	`								`
`4 + 4	l`	`								`

Different representations of a multiplication task:

Programs 5.3.2: Ten times ten	Results
`10 * 10`	100
`Xl:= 1 ..10` `Yl:=	*l 10 at X`	` 1									|` ` 2									|` ` 3									|` ` 4									|` ` 5									|` ` 6									|` ` 7									|` ` 8									|` ` 9									|` `10									|`

`10 * 10 |l`
`++text`
`cut 10`
	`||||||||||`
	`||||||||||`
	`||||||||||`
	`||||||||||`
	`||||||||||`
	`||||||||||`
	`||||||||||`

	|||||||||| |||||||||| ||||||||||

Programs 5.3.3: Two representations of a subtraction task	Result
`10 - 5`	`5`
`[ | | | | | | | | | | ] - 5`	`| | | | |`

Program 5.3.4: Place next to each number smaller than 16 the corresponding stroke list	Result

`Xl:=0 ..15`
`Y :=X |l`
	`X,Y l`
	`0`
	`1 |`
	`2 | |`
	`3 | | |`
	`4 | | | |`
	`5 | | | | |`
	`6 | | | | | |`
	`7 | | | | | | |`
	`8 | | | | | | | |`
	`9 | | | | | | | | |`
	`10 | | | | | | | | | |`
	`11 | | | | | | | | | | |`
	`12 | | | | | | | | | | | |`
	`13 | | | | | | | | | | | | |`
	`14 | | | | | | | | | | | | | |`
	`15 | | | | | | | | | | | | | | |`

5.4 o++o programs to kindergarten?

Which of the following programs are useful for understanding and which are teachable? When is a syntax too incomprehensible? These questions are outlined below.

Multiplication is counting the number of strokes in a rectangle?

Program 5.4.1: Each of four children gets 3 apples. How many apples are there in total ?	Result (tabh)

`NAMEl := Ernst Clara Sophia Claudia`
`APPLEl:= [| | |] at NAME`
`++`
	Intermediate result after the first 2 lines
	`NAME,   APPLE l`
	`Ernst   | | |`
	`Clara   | | |`
	`Sophia  | | |`
	`Claudia | | |`
	Final result (++ stands for many additions)
	`12`

The last line can also be replaced and you get the same result.

Program 5.4.2: Each of four children gets 3 apples. How many apples are there in total?	Result (tabh)

`NAMEl := Ernst Clara Sophia Claudia`
`APPLEl:= [| | |] at NAME`
`gib APPLEl`
`++1`
	Intermediate result after the first 2 lines
	`NAME,   APPLE l`
	`Ernst   | | |`
	`Clara   | | |`
	`Sophia  | | |`

	`Claudia ❘ ❘ ❘`
	Final result (++1 counts)
	12

Program 5.4.3: Counting different kinds of animals

```
ANIMAL:=elephant deer elephant pig elephant deer pig pig elephant
gib ANIMAL,CNT m   CNT:=ANIMAL!++|
```

Result (tabh output):

```
ANIMAL,  CNT l
Elephant | | | |
Deer     | |
Pig      | | |
```

Program 5.4.4: Counting cars

```
<TAB!
BRAND,COLOR,  TYPE, WEIGHT l
VW    Blue    Polo   1250
IFA   Papyrus 500     580
VW    Blue    Golf   1450
Audi  Yellow  Quatro 2070
VW    Blue    Polo   1380
IFA   Beige   601     620
VW    Red     Golf   1400
Audi  Red     Quatro 2100
IFA   Beige   601     620
VW    Beige   Polo   1300
!TAB>
gib BRAND,CNT,(COLOR,CNT m) m
    CNT:=TYPE ! ++|
```

Result (tabh)

```
BRAND,CNT,     (COLOR,  CNT2 l) l
Audi  | |        Yellow  |
                 Red     |
IFA   | | |      Beige   | |
                 Papyrus |
VW    | | | | |  Beige   |
                 Blue    | | |
                 Red     |
```

Programs 5.4.5: 3 Division Operations	Results
13 div 4	3
13 : 4	3.25
13 divrest 4	3,1

All these operations seem too complicated for kindergarten.
If one calculates not only with numbers but also with tables, one could introduce new division operations. However, this cannot be discussed to the end at this point.

Program 5.4.6: Problem: Distribute 15 apples among 4 children. Who designs the o++o program?	Result (tabh)

	Ernst			| Clara			| Sophia			| Claudia			

Another very important operation of digitization is selection. Would a database operation like selection be teachable to some degree?

given:

NAME,	AGE l
Ernst	8
Clara	6
Sophia	6
Claudia	4
Ulrike	5
Käthe	4

myfamily.tab

Program 5.4.7: How old is Claudia?	Result
`aus myfamily.tab` `avec Claudia`	NAME, AGE l Claudia 4

avec is French and means with

Program 5.4.8: All 6 year old children are wanted	Result
`aus myfamily.tab` `avec AGE = 6`	NAME, AGE l Clara 6 Sophia 6

Program 5.4.9: All children younger than 6 are wanted	Result
`myfamily.tab` `avec AGE < 6`	NAME, AGE l Claudia 4 Ulrike 5 Käthe 4

6 o++o in School Lessons

There are many possible applications for o++o in school. Especially in the subjects mathematics and computer science. But also in all other subjects o++o can be used to extract data from given tables, documents or from Wikipedia (see: chapter 11). We do not want to present all possible typical query examples here. We want to limit ourselves to the so-called "brute force algorithms" for mathematics. These are the simplest, i.e., the methodologically best. Since all these algorithms are implemented in main memory, we need not worry about efficiency. Now we start with a simple algorithm. We hope that it is the simplest program for a zero. The section ends with programs for grading students and considerations that may be important for kindergarten.

Program 6.1: Calculate in a simple way the zero of the sine function in the interval [3, 4].	Result

`Xl:= 3 ... 4! 0.00001`
`avec X sin <0`
`avec X pos =1`
	`Xl`
	`3.1416`

Program 6.2: Calculate in a simple way the zero of the sine function in the interval [3, 4].	Result
`Xl:= 3 ... 4! 0.00001` `avec X sin * (X +0.00001 sin) <=0`	`Xl` `3.14159`

Program 6.3: Calculate the integer zeros of the polynomial "X^2 -15X+56".	Result (tabh)

`Xl:= -100 .. 100`
`avec X poly [1 -15 56]=0`
`# or avec X- 15 *X + 56=0`
	`Xl`
	`7 8`

With the following programs, it is shown that students who have not learned integral and differential calculus are nevertheless able to understand and use their school applications, which are essentially:

1. How large are areas under curves?
2. What are local extrema of functions?

Program 6.4: Calculate the area under a circular arc with diameter 4 in the interval [0, 2].	Result

`Xl:= 0 ... 2!0.0001`
`HEIGHT:= X*X - 4 abs sqrt`
`RECTANGLE:=HEIGHT*0.0001`
`++ RECTANGLE`
	`AGG`
	`3.14169223791`

Program 6.5: Query 6.4, but shorter and more precise.

```
PI:=0 ... 2!0.000'001 poly [-1 0 4] sqrt*0.000'001 ++ '3
```

Result

```
PI
3.141'593'653'28
```

Program 6.6: Determine pi by zero determination with interval bisection

```
MI,LE,RI l:= 1.,2.,4.  while RI - LE >= 0.000'000'01 !
```

```
             LE pred +(RI pred) :2;
             (MI,RI pred) if MI sin > 0!(LE pred,MI)
avec MI pos- =1
gib MI
'3
```

Intermediate result after first program line (tab):

```
MI            ,LE           ,RI          l
1.             2.            4.
3.             3.            4.
3.5            3.            3.5
3.25           3.            3.25
3.125          3.125         3.25
3.1875         3.125         3.1875
3.15625        3.125         3.15625
3.140625       3.140625      3.15625
3.1484375      3.140625      3.1484375
3.14453125     3.140625      3.14453125
3.142578125    3.140625      3.142578125
3.1416015625   3.140625      3.1416015625
3.14111328125 3.14111328125 3.1416015625
3.14135742188 3.14135742188 3.1416015625
3.14147949219 3.14147949219 3.1416015625
3.14154052734 3.14154052734 3.1416015625
3.14157104492 3.14157104492 3.1416015625
3.14158630371 3.14158630371 3.1416015625
3.14159393311 3.14158630371 3.14159393311
3.14159011841 3.14159011841 3.14159393311
3.14159202576 3.14159202576 3.14159393311
3.14159297943 3.14159202576 3.14159297943
3.14159250259 3.14159250259 3.14159297943
3.14159274101 3.14159250259 3.14159274101
3.1415926218  3.1415926218  3.14159274101
3.14159268141 3.1415926218  3.14159268141
3.14159265161 3.14159265161 3.14159268141
3.14159266651 3.14159265161 3.14159266651
```

Result

```
MI
3.141'592'666'51
```

It can be seen that the first 7 digits of pi (3.141'592'65359...) after the decimal point are correct and that this correctness requires only 28 steps.

Program 6.7: Calculate the first 7 Fibonacci numbers.

```
Xl:= 1 .. 7
FIB1,FIB2:= 0,1 for FIB2 pred ;preds ++ at X
```

Result (tab)

```
X, FIB1, FIB2 l
1  0       1
2  1       1
3  1       2
4  2       3
5  3       5
```

```
6   5       8
7   8      13
```

Program 6.8: Calculate the Pascal triangle up to the exponent 9.

```
Nl:= 0 .. 9
XTUP:= 1 for 0,(XTUP pred) + (XTUP pred, 0) at N
wort
```

Result (tab)

```
N, XTUP l
0  1
1  1,1
2  1,2,1
3  1,3,3,1
4  1,4,6,4,1
5  1,5,10,10,5,1
6  1,6,15,20,15,6,1
7  1,7,21,35,35,21,7,1
8  1,8,28,56,70,56,28,8,1
9  1,9,36,84,126,126,84,36,9,1
```

Now we present a brute force algorithm for a maximum.

Program 6.9: Find in a simple way for the local maximum of the sine function in the interval [1, 3].

```
LOCMAX:=1 ... 3.!0.00001 sin max '3
```

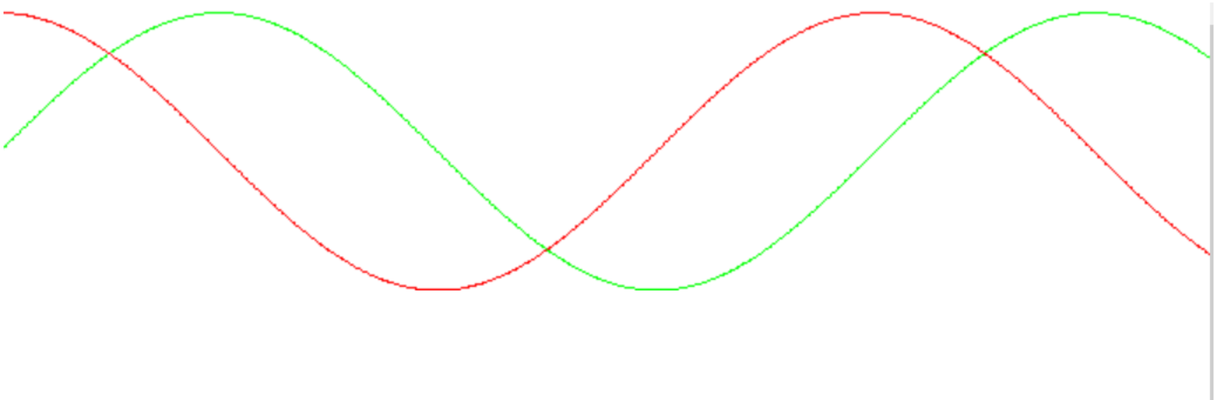
Result

```
LOCMAX
0.999'999'999'993
```

Program 6.10: Calculate the sine function and an approximation of the first derivative in the interval [0,4].

```
Xl:= 0 ... 10!0.01
SINUS := X sin
DERIVATIVE:=X+0.000'1 sin -(X sin):0.000'1
RGBSIN:=green leftat SINUS
RGBDERIVATIVE:=red leftat DERIVATIVE
```

Result (bild):



Result (tab): It consists of 1001 lines.

```
X     ,RGBSIN   ,SINUS               ,RGBDERIVATIVE  ,DERIVATIVE        l
 0.    0.,1.,0.  0.                   1.,0.,0.         0.999999998333
 0.01 0.,1.,0.  0.00999983333417 1.,0.,0.         0.999949498758
 0.02 0.,1.,0.  0.0199986666933  1.,0.,0.         0.999799005067
 0.03 0.,1.,0.  0.0299955002025  1.,0.,0.         0.999548532308
 0.04 0.,1.,0.  0.0399893341866  1.,0.,0.         0.999198105529
 0.05 0.,1.,0.  0.0499791692707  1.,0.,0.         0.998747759772
 0.06 0.,1.,0.  0.0599640064794  1.,0.,0.         0.998197540071
 0.07 0.,1.,0.  0.0699428473375  1.,0.,0.         0.997547501448
 0.08 0.,1.,0.  0.0799146939692  1.,0.,0.         0.996797708907
 0.09 0.,1.,0.  0.089878549198   1.,0.,0.         0.995948237425
 0.1  0.,1.,0.  0.0998334166468  1.,0.,0.         0.994999171949
...
 9.98 0.,1.,0. -0.527131998452   1.,0.,0.        -0.849757059217
 9.99 0.,1.,0. -0.535603334614   1.,0.,0.        -0.844442914713
10.   0.,1.,0. -0.544021110889   1.,0.,0.        -0.83904432662
```

The following examples are based on a fictitious table of grades with exams:

```
NAME,       (SUBJECT,    EXAl,MARKl l)l
Einstein     German      1 3  1 2 1 3 1
             Physics     1 a  1 2 1 1 1 1
             Algebra     1 2  1 1 2
             Art         3 3  2 1
Gauss        German      2 3  1 2
             Algebra     1 1  1 1 1
Guericke     physics     s 1  1 2 1
             German      2 1  1 2 1 1
             Algebra     1 1  2 1 1 2
Newton       Physics     1 1  2 1 1
Confucius    Philosophy 1 1  1 1 1 1
             Chinese     1 1  2 1 1
Marx         economics   1 1  2 1 2 1
             Philosophy 1 2  1 3 1
Brecht       German      1 1  1 1 1 1 1 1 1 1
             Philosophy 1 2  2 1 1 2
Cantor       set theory 1 1  1 1 1
```

Tabment 6.1: guys.tabh (s: sick, a: absent)

Program 6.11: Calculate the weighted average scores for each person and subject and the total value. Sort the data.

```
aus guys.tabh
gib AVG,(NAME,AVG,(SUBJECT,AVG m)m)
    AVG:=EXAl ++: *0.6 + (MARKl ++: *0.4)! ++:
rnd 1
```

Result (tab):

```
AVG  ,(NAME      ,AVG2  ,(SUBJECT    ,AVG3   m) m)
1.4    Brecht     1.3     German      1.0
                          Philosophy 1.5
       Cantor     1.0     set theory 1.0
       Confucius 1.1     Chinese     1.1
                          Philosophy 1.0
       Einstein   1.7     Algebra     1.4
                          Art         2.4
```

```
                German      1.8
                Physics     1.1
Gauss     1.6   Algebra     1.0
                German      2.1
Guericke  1.2   Algebra     1.2
                German      1.4
                physics     1.1
Marx      1.4   Philosophy  1.6
                economics   1.2
Newton    1.1   Physics     1.1
```

If we want to calculate the average after the completion of the first test, we can use the following formula:

```
AVG:= EXAl nth 1 *0.3 + (MARKl ++: *0.7)! ++:
```

Program 6.12: Determine all subjects and individuals that received a 1 and a subsequent 3.

```
aus  guys.tabh
avec NAME SUBJECT! MARK=1 & MARK succ=3 #succ = successor
```

Result:

```
NAME,    (SUBJECT,    EXAl, MARKl l)l
Einstein German       1 3    1 2 1 3 1
Marx     Philosophy 1 2    1 3 1
```

Now we turn to other "simple" problems. It may be that these tasks are important not only for school, but also for kindergarten. Now it is commonly assumed that addition of natural numbers is the easiest and division the most difficult of the four basic arithmetic operations. This could be wrong. We did experiments with a 3-year-old and a 6-year-old kindergartener. The task was to divide 11 apples among four children. The four children were represented by photographs. Neither the 6-year-old child nor the 3-year-old child had a problem. They obtained the same result in the division. It was presented in a table:

```
CHILD,  APPLE
Ernst   | | |
Clara   | | |
Sophia  | | |
Claudia | |
```

Tabment 6.1: 11 divided by 4

What can we learn from this experiment?

1. Young children can not divide an apple. They do not yet have a clear understanding of 1/2 or 2/3, ..., so that "ordinary" division cannot be taught.
2. There is no remainder in the division; there is no reason to waste anything.

Let's consider **addition**, the next simplest operation. The simplest representation of the number three are three strokes. The same is true for any number of other natural numbers. Here we consider only two numbers: 3 and 4.
We have to represent them by lists or bags (multisets) because the set {| |} is the same as {|}. The result of each operation would be one.

APPLE l
| | |

three.tabh

APPLE l
| | | |

four.tabh

Program 6.13: three plus four	Result
`aus three.tabh,four.tabh` `gib APPLEl`	APPLEl | | | | | | |

Here, too, it becomes clear that such a process could require only a small amount of effort in the classroom. But what is the result of 4 apples and 3 pears? Since each pair of two tabments is again a tabment, the result of this "addition" could be a tabment of the type APPLEl,PEARl.

Multiplication can also be handled in a very simple way. Consider the very simple question of how many apples are needed for 4 children if each child wants 3 apples:

Program 6.14: four times 3	Intermediate result after line 2

`CHILDl:= Ernst Clara Sophia Claudia`
`APPLEl:= [ | | | ] at CHILD`
`gib APPLEl`
	CHILD, APPLE l
	Ernst | | |
	Clara | | |
	Sophia | | |
	Claudia | | |

Not only is this multiplication algorithm simpler, it also makes it clear that multiplication is essentially calculating the area of a rectangle.
We also obtain the above intermediate result by the following program:

```
CHILDl:=Ernst Clara Sophia Claudia
APPLES:= [ | | | ] at CHILD
gib CHILD,APPLES l # APPLES is atomic, i.e. each apple list is transferred
                   # as one unit
```

If we want to apply the **subtraction operation** to collections with different elements (sets), the subtraction can be expressed by a selection.

Program 6.15: Subtraction (difference) with sets	Result
`NAMEm := {Ernst Clara Ulrike}` `sans NAME in {Ulrike Sophia}`	NAMEl Clara Ernst

We conclude this section with the following statements:

1. The result of our "arithmetic operations" are not numbers, but tables.
2. Dealing with tables is probably easier than dealing with numbers because the level of abstraction is lower.

7 Multiplication, School and Digitization

In this chapter, it will be shown that the common multiplication algorithm for decimal numbers could and should be supplemented by simpler ones and that, more generally, deep digitization (TD) should be pursued. Deep digitization can probably only be implemented through mathematical understanding. Unlike shallow digitization, where the user is usually presented with a computer result by simply clicking a button and is often unsure if that result is correct, deep digitization should allow the user to understand the result in the same way as calculating 132.66 times 453.2 with a calculator. That could be the area of his property. The big difference between today's use of calculators and today's use of powerful computers is that users have spent years learning the single data operations: `+ − * : sin log ... .` Bulk data operations are not yet on the curriculum. Selection - sometimes called a filter operation - and operations to merge table contents for restructuring ... we count among the mass data operations. These are not applied to individual numbers, but to possibly very large structured tables that may contain words and text in addition to numbers. If the user has understood such mass data operations and they have been implemented within the framework of a programming language, he can also interpret these results and, in case of doubt, correct, change or improve them.

7.1 Who can multiply in their head?

Incident 1

In a mathematics exam that a second-year pharmacy student from Bologna had to take, the student had to calculate 7 times 8, among other things. The pharmacy student: 59
The algebra professor: But 59 is not an even number. The pharmacy student: 64

Incident 2

Wallerie - an Erfurt kindergarten girl in the large group - is already a student today.
I gave her a task: How many effervescent bottles does a crate with 4 rows contain if there are 5 bottles in each row?
Wallerie thought for a while: Nineteen
Her father - a young engineer: You don't calculate, you guess.

Incident 3

I ask Isabella, a second grade pupil from Gerwisch: How much is 3 times 4? After a while: Twelve
The father: That took a long time.

From the second occurrence, I conclude that preschoolers have already understood the essence of multiplication. Of course, it is possible that some preschoolers.... cannot calculate 4 times 5 exactly in their head. But 3 times 4 I would trust any child to do. There is no question that they will never be able to calculate 12 times 13 in this way. In fact, I don't think any human being is capable of calculating 7 times 8 in their head. Older adults have had to calculate(?) the multiplication tables so many times in school that they can only do it by heart and don't remember how they multiplied as a child. It used to be very important to know the multiplication table by heart because it was a prerequisite for written decimal multiplication.
The opinion of an amateur neurologist: In the many school years that the multiplication tables were taught, the original neuron connections or brain cells were "overwritten" and are practically no longer present.

Therefore, the vast majority of adults are not able to perform the original multiplication in their heads. They can only do multiplication tables by heart and cannot do written decimal multiplication in their heads. Even when multiplying smaller numbers such as 29 times 63, they will work with easier-to-use arithmetic laws and not use the algorithm in which their teachers, parents and grandparents invested a lot of time and effort. Based on incidents 1, 2, and 3, one can even surmise that many adults don't even know that children have to do math to get the results. When you're that young, you can't memorize it yet without doing the math. An almost correct answer indicates that arithmetic has been done, just as a very quick answer indicates that arithmetic has not been done, and thus no thought has been given.

7.2 Who can multiply in writing?

Calculating `7 times 8` with a pencil should be mastered by every child in the second grade. The prerequisite is that you can imagine the numbers up to one hundred. You can do that if you can count to a hundred. If you illustrate the task, many children should be able to solve it even faster:
Each of the seven children wants eight candies. How many candies do you need to buy?

1. Write the names of seven children one below the other.
2. Put eight strokes legibly after each name.
3. Count all the strokes.

Everyone can imagine that you can multiply arbitrarily large numbers with this algorithm. But it would be nice if you could not only pronounce the result and write it as a word, but if you could write the result more compactly as a decimal number.

4. Convert the result into a decimal number.

If someone wants to calculate `100 times 100 in` this way, the probability of getting a correct result is very low. Moreover, it would take a very, very long time. In the age of powerful computers, however, these arguments should be insignificant. What matters is to have a clear understanding of an algorithm. The question remains:
Is stroke list multiplication the simplest multiplication algorithm?

7.3 Who can program the multiplication?

Incident 4

An engineer from my former institute tells me that she was the first to learn assembler at the Staßfurt television factory. With it, she was able to solve efficiency problems after the reunification, which a new boss of the TV set factory had not trusted her to do. She remembers very clearly that she had great difficulty learning C.

Incident 5

Since I was not a professor, I was able to participate in IBM Germany's visiting scientist program in 1992. In the IBM Research Center - the scientific center in Heidelberg - the database project AIMP (Advanced Information Management Prototype) had been developed for many years. Essentially, this involved the database query language HDBL (Heidelberg Database Language), with which NF^2 relations could be processed. These relations generalize the table concept of the relational data model to structured tables. In the end, however, IBM Germany was not able to convince the headquarters in the USA that their prototype should be brought to market. Even more remarkable to me was that an employee who was listed as an author in a very large number of publications on HDBL could not answer the simplest questions about HDBL. He then explained to me that he had programmed for years in PASCAL for HDBL file management and actually had no interest in formulating queries in HDBL.

Incident 6

I wanted to understand UNIX and bought the book "UNIX und C" from VEB Verlag und Technik. I read, marked, read, marked, read, marked and repeated. But only slowly and with little success.

In my opinion, the most important conclusion from these events is that it is very important which language you learn first. Relearning should usually be more difficult than newlearning. Furthermore, one must assume that ways of thinking from procedural or object-oriented programming languages offer few advantages for query languages. Furthermore, one can certainly not learn programming by reading alone. You have to make mistakes yourself.
I think operations can be taught to most people through algorithms rather than descriptive formalizations. If someone can program an algorithm, he can or should break it down into more elementary steps to understand it better. If suitable programming languages are available, decimal multiplication does not stand a chance against the stroke list algorithm in terms of learnability and readability. No matter how high the abstraction level of a programming language is. I believe less than one percent of the world's population today can program decimal multiplication in 30 minutes. In fact, far less than 1 percent of the world's population are software developers.

Recently, according to Hacker Rank, China not only took first place in functional programming through well-organized programming Olympiads, but also first place in the overall ranking of the "best developers" in the world, ahead of Russia. For me personally, this is particularly impressive because a Chinese student who graduated with me in 2009 told me that he left China because he was supposed to learn PASCAL at his Chinese university first. He did not consider this (methodically perhaps not so bad) language to be up to date.
The economic effects, which are certainly also due to China's education policy, can already be clearly seen today. From 1990 to today, Germany's share of global exports of high-tech goods has almost halved. China has increased its share from one to 24 percent during this time (according to Handelsblatt).

The corresponding code can certainly be solved most elegantly in the functional French language OCaml. That OCaml has very good concepts and efficient implementations can perhaps also be seen from the fact that Microsoft has copied OCaml. F# even has the same syntax as OCaml.
Let's first look at a multiplication algorithm at what I consider the high abstraction level of functional programming.

However, let's first briefly clarify how multiplication is practically performed in OCaml:

Program 7.3.1: User-level integer multiplication in OCaml	Result
`7*8;;`	`- : int = 56`

For 7.1 times 8.1 you have to choose another operation symbol.

Program 7.3.2: Multiplication of floating point numbers on user level in OCaml	Result
`7.1 *. 8.1;;`	`- : float = 57.51`

OCaml also has a data type `bigint` for arbitrarily large integers.
Certainly, one can imagine that even very small children can click the keys `7`, `*`, `8` and `=`. But it should be clear that this clicking will not lead to understanding, nor will these clickers themselves be able to solve a corresponding problem. Even typing such examples a hundred times will not sufficiently improve understanding of multiplication. Now follows a program for multiplication in OCaml. We call the corresponding operation `mult`. This is based on a previously defined new data type `nat`. Without the "auxiliary" operation `add`, it might be difficult to program `mult`. A conversion of the self-defined datatype into decimal numbers is omitted at this point. The following syntax is very elegant, but in a certain sense also tricky. We do not want to go into details here and refer to the OCaml documentation on the Internet. Even if you are not familiar with functional programming and with OCaml, you can see that the programs are ingeniously compact and clear. The

reason for this is that in this case you don't have to think of a natural number in decimal or binary. Rather, the definition of `nat is` based on the childish idea of counting `zero one two three` ... except that you don't always have to invent new words for new numbers. According to the following definition, 0=zero, 1=successor zero, 2=successor(successor zero), and 3=successor(successor zero)), ... This naive conception of the natural numbers, which certainly every child already possesses very early without learning it, was "rediscovered" only in 1889 by Peano.
A side note: Picasso actually also worked his whole life to be able to paint in a childlike, naive way again.
All **axioms of Peano** are hidden in the two lines of the program. They actually only say:

1. Zero is a natural number
2. Every natural number has exactly one successor.

The fact that the operations are practically useless without the use of decimal numbers for input and output is irrelevant at this point. Here it is only about getting an impression of the logical problems.

Program 7.3.3: Multiplication of natural numbers with own data type in OCaml

```
type nat =
   | Zero
   | Succ of nat
;;
let rec add x = function
   | zero   -> x
   | Succ y -> Nachf(add x y)
;;
let rec mult x = function
   | zero   -> zero
   | Succ y -> add x (mult x y)
  ;;
let seven=Succ(Succ(Succ(Succ(Succ(Succ(Succ Zero))))));;
let eight=Succ seven;;
mult seven eight;;
```

Result (nat):

```
- : nat =
Succ
 (Succ
   (Succ
     (Succ
       (Succ
         (Succ
           (Succ
             (Succ
               (Succ
                 (Succ
                   (Succ
                     (Succ
                       (Succ
                         (Succ
                           (Succ
                             (Succ
                               (Succ
                                 (Succ
                                   (Succ
                                     (Succ
                                       (Succ
                                         (Succ
                                           (Succ
                                             (Succ
                                               (Succ
                                                 (Succ
```

```
                                                      (Succ
                                                        (Succ
                                                          (Succ
                                                            (Succ
                                                              (Succ
                                                                (Succ
                                                                  (Succ
                                                                    (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     (Succ
                                                                     Zero))))))))
))))))))))))))))))))))))))))))))))))))))))))))))
```

I would be very happy if someone would send me another well understandable own multiplication program. It will then be published on the **ottops.**de page.

Program 7.3.4: Stroke list multiplication in o++o

```
NAMEl:=Tina Ernst Clara Sophia Ulrike Claudia Kathe
STROKEl:= [| | | | | | | | ] at NAME
gib STROKEl
++l
```

Intermediate result after 2 lines in tabh format

```
NAME,   STROKEl  l
Tina    | | | | | | | |
Ernst   | | | | | | | |
Clara   | | | | | | | |
Sophia  | | | | | | | |
Ulrike  | | | | | | | |
Claudia | | | | | | | |
Kathe   | | | | | | | |
```

Final result

```
56
```

If you are only interested in the end result, you can also replace all names with one. You could also write a gib statement for the last two lines that counts through regardless of the given structure:

Program 7.3.5: Shortened stroke list multiplication in o++o

```
NAMEl  := otto *l 7
STROKEl:= | *l 8 at NAME
```

gib SEVEN_TIME_EIGHT   SEVEN_TIMES_EIGHT:= STROKE! ++1
Result
SEVEN_TIMES_EIGHT 56

Everyone can freely choose new column names in o++o. However, lowercase letters must not be used. Instead of SEVEN_TIMES_EIGHT you could also choose the shorter PRODUCT_7_8. The column name SEVEN_TIMES_EIGHT in the gib part does not occur in the table created with the first two rows. It is intended as a new aggregation column. The last line expresses that this single output column should be calculated by the counting aggregation (++1). Because of the indentation (4 spaces) this line logically still belongs to the gib statement.
This program can only be formulated because o++o works with structured tables. Everybody should judge for himself which multiplication is more child-like and therefore easier to understand. At this point it should be mentioned that o++o multiplication is much more general than in other programming languages. However, this does not mean the program above, but the operation hidden behind the symbol *. For example, you can multiply a whole tabment by a number:

Program 7.3.6: Convert several German net prices with o++o into gross prices.
66.1 675.8 77 *1.19
Result (tabh)
PZAHL1 78.659 804.202 91.63

A math teacher at the Magdeburg Physical Education High School noted that students have no trouble multiplying in writing in positional systems other than the decimal system. Let's talk about binary numbers for a moment. From my point of view, it is very interesting that it again took a mathematical genius like Leibniz to make dual numbers respectable, although calculating with dual numbers is much easier than calculating with decimal numbers. This would, of course, also make binary multiplication much easier to teach than decimal multiplication. One only has to memorize a small multiplication table:

- 0*0=0
- 1*0=0
- 0*1=0
- 1*1=1

This algorithm has other advantages: It is very efficient - both in terms of memory and speed. That is why it is used in computers. Not everyone needs to be aware of it. Even if the user knows only decimal multiplication, there can be no problems. His decimal numbers can be converted to binary without his knowledge, binary calculations are performed, and when the result is output, it is converted back to decimal. Similarly, one can imagine that a decimal number is converted to a dash number for a child (user), dash multiplication, which he should have understood, is applied, and the dash number is converted back to decimal. This is to express that the user should learn the most memorable and useful multiplication algorithm and be able to trust that the computer experts will "implement" his algorithm correctly.
This is only to say that the results should be correct. Internally, another, more efficient algorithm can be used. Of course, this idea of internal optimization does not only apply to multiplication algorithms.
The section concludes with a multiplication algorithm that is reminiscent of pivot tables, but very close to the decimal multiplication algorithm taught in schools today.
The pivot element can be determined with 2 o++o lines.

Program 7.3.7: Multiplication of a list of length 2 by a triple using matrix multiplication in o++o
10 3 *mat (100,20,4) SU:=ZAHL tup ++ addaggs ZAHL ! ++
Intermediate result after the first line in tab format
`ZAHL ,ZAHL ,ZAHL  l`
`1000  200   40` ` 300   60   12`
Intermediate result after applying the second line in tab format:
`ZAHL ,ZAHL ,ZAHL ,SU  l`
`1000  200   40    1240` ` 300   60   12     372`
Final result in tab format
`ZAHL ,ZAHL ,ZAHL ,SU  l`

`1000  200   40    1240`
` 300   60   12     372`
`1300  260   52    1612`

The final result of multiplying `13 times 124` is `1612.`
Math teachers must find out whether this multiplication is easier to teach than today's multiplication with decimals by having entire classes multiply using both methods.

Program 7.3.8: A complete o++o programm for multiplication, based on matrix-multiplication

```
X:=4321
Y:=678
XXl:=X zil zahl
YYl:=Y zil zahl
XL:=XXl ++1
YL:=YYl ++1
BXl:= 10 *l XL hoch (XL- 1 ... 0! -1) * XXl
BYl:= 10 *l YL hoch (YL- 1 ... 0! -1) * YYl
MATRIX:= BXl *mat (BYl transpose)
SU:=SP1 tup ++
addaggs SP1 ! ++
gib SU l
ultimo
'3
```

Result (tab)

```
SUl
2'929'638
```

The operation `zil` transfers a text into the list of its characters (Chinese zi). This and other operations could also be used advantageously for teaching German or English. Each letter is converted into a digit by the conversion function `zahl`. Teaching pivot multiplication in one form or another would also prepare students for the use of pivot tables, which play a large role in today's practice.

The last program in this section is intended to implement pivot multiplication with the help of o++o operations.

Program 7.3.9: More detailed matrix multiplication of factors 7'653 and 4'322 in o++o

```
defop $X myop.powerlist = begin
```

```
X1l:=$X zil zahl
gib X1 l-
XPOT:= (10 ^ (X1 pos - 1))
X2:= XPOT * X1
gib X2l-
end
aus 7'653 myop.powerlist
*mat 4'322 myop.powerlist transpose
addaggs SP1 ! ++
ultimo
++
'3
```

Final result (tab)

```
33'076'266
```

7.4 Stroke list multiplication versus decimal multiplication

The written decimal multiplication had a great importance because many people could calculate with it even two 8-digit numbers correctly with high probability. The correctness could be improved even more by the sample of nine. In addition, it was by far the fastest algorithm in earlier practice.
The later slide rule was faster, but not as accurate. The number table with logarithms was too demanding for some. In the age of computers, both techniques have already been mothballed.
The written decimal algorithm is still used by many people ..., and everyone who masters it is proud of his skills. The question is: Can we replace the decimal multiplication algorithm in school with the stroke list multiplication or/and complement it with other multiplication algorithms?
The dash multiplication has not only the advantage that it can be taught already in an earlier class. If this algorithm is repeated accordingly, everybody notices that just this multiplication realizes the standard application - a rectangular area calculation. To derive this application from decimal multiplication seems too difficult. Given the programmability of both algorithms, it should quickly become clear that dash multiplication is far superior to decimal multiplication. The dash list multiplication above requires 4 steps to be processed in sequence. In particular, there is no loop and no recursion. On the other hand, the above algorithm shows that a programming language must be able to work with structured tables in order to provide user-friendly multiplication programs. Since this stroke multiplication processes mass data in a sense, it prepares better for the digitization of society than the algorithm taught in schools today. If it is desired that everyone should be able to program multiplication, simpler multiplication algorithms must be taught.

7.5 How could enrich o++o the school curriculum?

We have developed a data model with an associated programming language o++o, which should not only be the basis for information systems for business, but also offer many advantages for school teaching. Since the language o++o is based on mathematical concepts, it should be integrated into mathematics classes. But, also in the other subjects o++o can be used usefully, because the extraction and visualization of information from the German Wikipedia seems to be important for every school subject. A programming language should enable students to better solve the many tasks they will face in their future. This is especially true for the digitalization ahead of us. Digital actually means that everything comes down to two things - zero and one. I can't imagine anyone tracing the powerful stroke list operation behind the gib statement, for example, to such thinking. Our axioms of the stroke list operation were formulated at a high abstract algebraic level, where thinking in terms of zeros and ones is only a hindrance. For mathematics, abstraction is more important than digitization in the true sense of the word. Although o++o has so far only dealt with questions of content, CSS in o++o can also be used to realize many questions of format. The following example shows that it also makes sense to capture form questions directly in o++o:

Program 7.5.1: The product of 5 numbers in thousands format in o++o
`28'911 5'233 199 6'311 6'781 ** '3`
Result
`1'288'424'128'758'129'267`

o++o could perhaps be taught in the lower grades:

Program 7.5.2: The sum of the first hundred numbers (Gauss problem)
`1 .. 100 ++`
Result
`5050`

I think that these and many other difficult problems could be taught in the lower grades as well. So as not to be misunderstood at this point. We should already be drawing on experience gained decades ago with calculators when introducing digitization in schools. Also, the too early and too wide use of calculators has probably led to many students having a poorer command of basic arithmetic..., worse at calculating in their heads than in earlier grades. With incorrect inputs to the calculator, many also seem unable to estimate the expected magnitudes of the results. For this reason, the calculator is not allowed here until seventh grade.

For example, I even think that spelling programs like WORD should not be taught in school until about the seventh grade, either. If a student has experienced firsthand that WORD corrects almost all of his spelling mistakes, it is very difficult to make him understand ... that his own spelling skills are important for his future. Similarly, I would have the introduction of digital whiteboards critically examined.

Last December, at the University of Halle, I noticed that mathematics professors were still working with blackboards and ordinary chalk.

Program 7.5.3: Execute an o++o program in the second class on the blackboard.
`4 3 1 12 ++`

This calculation on the blackboard with chalk or pencil could also prepare for future digitization. In addition, for motivation reasons, the teacher could already demonstrate to the lower grade students that the symbol `++ can be` used to solve the Gaussian problem or even larger problems. In my opinion, many people do not actively know how to formulate a conditional, although it is not difficult. However, it is not part of the curriculum. The conditions that select all people living in Magdeburg `LOCATION=Magdeburg` or filter out all rivers that are longer than 1000 km `LENGTH>1000` do not look complicated. Many can't do that, because today's search engines don't ask for that or can't handle it. But if I need to spontaneously extract important information from a company database in a future company, I need to know that.

In my opinion, students' problem-solving skills can be improved in many ways. Even the applications of differential and integral calculus could be taught in secondary schools without having to understand the difficult theories of Leibniz and Newton. With o++o, we can calculate areas under curves in a short line of code without using hard-to-read loops. An approximation of the area under a part of the first sinusoidal arc can be calculated in one line using Archimedes' 2000 year old algorithm:

Program 7.5.4: This o++o program does not require integral calculus!
`1 ... 2!0.0001 sin *0.0001 ++`
Result

```
0.956536680039
```

In the following, an application of differential calculus is presented, which can be performed without knowledge of differential calculus.

Program 7.5.5: An o++o program to approximate the local minimum of the parabola (a special polynomial) "3 x^2 + 4 x + 6"!

```
-10 ... 10!0.0001 poly [3 4 6] min
```

Result

```
4.66666667
```

I believe that this can be taught already in every 9th or 10th grade, without going into details here. When I talk to students, I sometimes have the impression that computer science classes are more about form issues (HTML, ...) than content. We know that it is very hard, but we should still reach the goal given by our Dr. A. Merkel: **Everyone should learn to read and calculate, but also to program**. If you look at programming languages like C, Java or Python, the goal is not feasible. For that, you need simpler languages that are able to solve end-user problems with short programs. C and Co. had other goals. They should serve to program systems on which hundreds or more people can work, which can contain many millions of lines of code and still work performantly. o++o follows the new paradigm of table-oriented programming and has above all the goal formulated by A. Merkel. If o++o had not put methodical and pragmatic questions in the foreground from the beginning, this goal would not be realizable also with o++o. Mastering operations for mass data seems to be necessary for a long-term digitization strategy.

7.6 Can the stroke list operation be taught as early as third grade?

As already mentioned, the gib statement, which includes the dash list operation, is a powerful tool. It can be used not only to sort normal flat tables, but also any tables. At the same time, you can also use aggregations such as `++` (sum), `++1` (count), etc. If third grade students have difficulty with a formal syntax, it does not necessarily mean that the algorithm behind it cannot be taught. For example, they could count animals. This does not have to be just a number. A table that determines the number for each type of animal would certainly be easy to teach as well:

Program 7.6.1: Counting animal species with strokes

```
ANIMAL:=donkey sow boar donkey boar sow donkey
gib ANIMAL,CNT m
    CNT:=ANIMAL! ++|
```

Result (tabh)

```
ANIMAL, CNT m
```

```
Boar    | |
Donkey  | | |
Sow     | |
```

The following example is a bit more demanding, because the results table is structured.

Program 7.6.2: Counting in structured tables

```
<TAB!
BRAND,COLOR,  TYPE,  WEIGHT l
VW    Blue    Polo   1250
IFA   Papyrus 500     580
VW    Blue    Golf   1450
Audi  Yellow  Quatro 2070
```

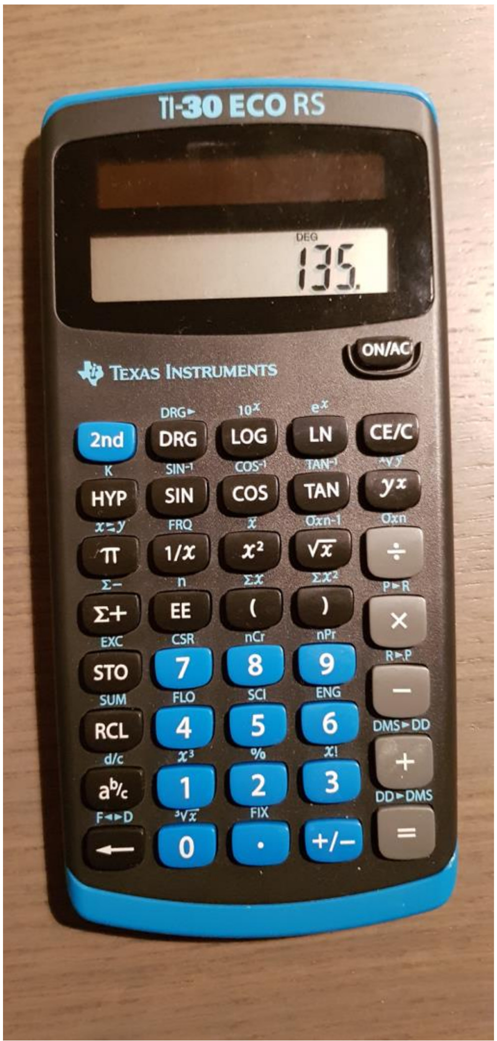
```
VW    Blue    Polo   1380
IFA   Beige   601     620
VW    Red     Golf   1400
Audi  Red     Quatro 2100
IFA   Beige   601     620
VW    Beige   Polo   1300
!TAB>
gib BRAND,CNT, (COLOR,CNT m) m
    CNT:=TYPE! ++|
```

Result:

```
BRAND, CNT,       (COLOR,  CNT2 m) m
Audi    | |        Yellow  |
                   Red     |
IFA     | | |      Beige   | |
                   Papyrus |
VW      | | | | |  Beige   |
                   Blue    | | |
                   Red     |
```

You can perhaps imagine children counting and sorting at the blackboard using this algorithm. In o++o a set (m) or a multiset (bag) is always sorted by the first column names. In the example above these are BRAND and COLOR.

That is, children can presumably sort data in structured tables. However, today's computer science students do not learn a sorting algorithm for structured tables. An article of mine in the German Wikipedia, which included especially this sorting, was deleted, because it "does not belong to the basic knowledge of a computer scientist".



7.7 Does the school calculator from Texas-Instruments calculate wrong?

The TI-30 ECO RS calculator shown on the left, which has been approved by German education ministries as a school pocket calculator, gives the following results for the task

2 hoch 2 hoch 3

64. Correct according to the rules of today's mathematical conventions, which can also be read in Wikipedia under operator order (right-associative), would be 256.

For hoch, however, you have to type the symbol y^x there.

Now, of course, you can say that every company can calculate as it pleases. They do that, too. With the Windows calculator (mode normal), 1 + 2 × 3 also results in a wrong solution in the sense of school mathematics. Saxony-Anhalt may not have enough money to sue the

American tech giant Microsoft. But how can we prevent many students from losing their orientation because of this "diversity"?

As the picture above suggests, the calculator makes a very good impression. However, it behaves differently from what is taught in school in many other aspects and is also difficult to use, making it prone to errors even in simple tasks.

In mathematics, the "sine of 3.14" is usually written as follows:

sin(3,14)

In the mathematics textbook "Schlüssel zur Mathematik" (Sekundarstufe Sachsen-Anhalt Klasse 10 Cornelsen, ISBN 978-3-06-0044558-7) it says more regrettably:

"The function f(x) = sin x is called a sine function."

The Texas Instruments calculator does not accept the comma as a decimal number separator and you must first press 3.14 and then the sin key. At least Texas Instruments is consistent in typing at this point. The square root of 4 is also found by first typing the 4 and then the square root sign. As everyone expects, the result is 2. But 2 without the decimal point would also be conceivable? With 2+2, Texas Instruments also determines 4. and not 4, although everyone knows that the result of this addition is an integer. To prevent misunderstandings at this point: We do not criticize that this Texas Instruments calculator chooses the more user-friendly typing variant for single-digit operations, but that curriculum and school practice differ substantially here.

Designations on the keyboard are also surprising.

Σ+ (EE, RCL, STO, ...).

Many people are already familiar with the M+ symbol - add to memory - due to predecessor computers. Is innovation to be feigned here?

In this context, it is also interesting to note that pocket calculators already existed in the 1970s whose range of functions was perfectly adequate for use in mathematics lessons and for a large number of applications, especially in the scientific and technical fields.

These calculators were characterized by a clear keyboard layout that did without multiple key assignments. The calculator architecture consistently implemented left-to-right arithmetic, and it was possible to dispense with bracket levels. The range of functions was limited to the necessary and frequently used functions. This minimized problems arising from different designs and ensured simple and intuitive operation.

One example is the scientific calculator shown in the figure, developed and produced in Japan in 1975.

From o++o point of view, however, the TI-30 ECO RS behaves correctly for the most part in these problems. For example, with 2 to the power of 3 to the power of 4, it chooses the way of calculating that the majority of people prefer, namely to calculate from left to right. This is also true for engineers, as I experienced many times. That one-digit functions are typed after the number (the argument), we also welcome, because this way of calculation also follows the principle from-left-to-right:

3.14 sin cos

The calculator from Texas-Instruments first calculates the sine and applies the cosine function to the result. This is not taught in math classes, but it is also easier to understand. Unfortunately, the calculator is not completely consistent at this point. At 1 + 2 × 3, it no longer calculates from-left-to-right. Now it calculates as Descartes supposedly wanted it to. Only so that one could write a polynomial somewhat more elegantly, humans gave up the general principle from-left-to-right to calculate. Since one can regard today also a list of numbers as input value, this argumentation from the 17-th century has no more right to exist from our view. Instead of

$X^3 + 2 * X^2 + 3 * X + 4$

we can today briefly and succinctly X poly 1 2 3 4 type.

In general, we also estimate that all of today's calculators are morally worn out. They should no longer be used at school at all. The first electronic, actually palm-sized calculator was developed as early as 1967 and had - as is still common today - a **very small display**. Since cell phones with much larger displays exist today in 2023 and we also know much more powerful apps with a much wider range of applications, calculators should generally be banned from school today or displayed in the school museum.

Let's consider a very simple problem. You want to add 10 numbers with the Texas Instruments calculator. At the end of the calculation, when you realize that the result cannot be correct, you cannot look at the input again. They have to type in all the numbers again. It is unclear whether they do this correctly if all these numbers consist of 10 digits.

Let's continue by looking at the % key. If you play with the calculator and type for example

10 %

the result is 0.1.

So you might suspect that the percent key is just mislabeled, and it just divides by 100. The percent key is also hard to type on this calculator, since you have to type 2nd beforehand. Also, once you find the little blue percent sign, which doesn't have its own key, you have to concentrate very hard to see if you should press the key above or below it. These are, of course, potential sources of error. Of course, you also have to know whether the 2nd key is only valid for the next operation or until I press it again. If one then types for example

10 + 10 %

If you press = , you first get 1. Only when you press = further does the current user get the number 11 that he probably wants.

But if you think mathematically, only one of the following two solutions comes into question:

10 + (10 %)

or

(10 + 10) %

You get 10.1 in the first case and 0.2 in the second.

That is, with this symbol mathematical thinking is contradicted. How should one understand

10 + 10 %

differently as a term? Why do all students need to learn a term definition if it is not applied in calculator practice at school?

To our knowledge, there is only one programming language that uses this symbol at all in connection with percentage calculation. Here, however, +% is used as a two-digit operation symbol. This also makes it mathematically clear and clean.

Just as the three letters of sin represent an operation symbol, +% is also an operation.

In o++o results in 10 +% 10 11. .

If you type in the Texas Instruments calculator

10 sin x^2

so you never see on the display which operation symbol you have just typed or typed before. Furthermore, the keyboard labels make it difficult to understand the "dot before dash" rule when the multiplication sign consists of 2 dashes and the division sign contains a dash. We conclude the section with what appears to be a very simple multiple addition. We think that hardly anyone can correctly manage an addition of very many numbers with a calculator.

Sophia has built a tiger out of Lego bricks. At the end of the description of the construction set, all the types of bricks used are listed with the number of bricks used. Grandpa wants to know how many Lego bricks the tiger is made of?

Program 7.7.1: Sum of many numbers

```
8 4 2 1 15 8 4 1 10 6 6 4
,4 1 1 2 14 4 2 2 4 1 2
,4 2 1 4 4 2 1 2 1 1 1 1
,6 2 1 2 1 2 4 2 2 1 4 1
,4 2 1 2 1 17 11 3 1 2 2
,2 3 4 4 4 11 2 1 2 1 2 4
,7 4 2 4 14 6 5 4 6 4 12 4
,2 2 2 1 6 1 1 2 2 1 1 2
,4 4 2 6 2 2 6 1 4 4 5 5 22
,4 14 4 8 5 8 4 1 6 12 6 2 4
,1 4 2 4 10 6 2 8 1 1 12 1
,1 2 1 1 8 6 6 6 2 1 1 4 4
,2 2 6 2 6 4 17 7 26 2 4 2
,8 12 2 6 6 4 16 8 6
,2 4 3 2 1 2 4
,2 2 1 1 1 2 1
++
```

Result:

```
755
```

The above program consists of a tuple of 16 lists of numbers. This is certainly more advantageous than typing a single list or tuple. In the latter case, one would have to type commas instead of the many spaces that are easy to type, which would certainly not be an advantage.

Since people often make typing errors or type one key too many or too few, the question remains: Is the result correct?

There is no sample of nine for addition. Here some possibilities for samples shall be presented. One could first check if the number of numbers in the program is correct.

Program 7.7.2: How many numbers does each list contain?

```
8 4 2 1 15 8 4 1 10 6 6 4
,4 1 1 2 14 4 2 2 4 1 2
,4 2 1 4 4 2 1 2 1 1 1 1
,6 2 1 2 1 2 4 2 2 1 4 1
,4 2 1 2 1 17 11 3 1 2 2
,2 3 4 4 4 11 2 1 2 1 2 4
,7 4 2 4 14 6 5 4 6 4 12 4
,2 2 2 1 6 1 1 2 2 1 1 2
,4 4 2 6 2 2 6 1 4 4 5 5 22
,4 14 4 8 5 8 4 1 6 12 6 2 4
,1 4 2 4 10 6 2 8 1 1 12 1
,1 2 1 1 8 6 6 6 2 1 1 4 4
```

```
,2 2 6 2 6 4 17 7 26 2 4 2
,8 12 2 6 6 4 16 8 6
,2 4 3 2 1 2 4
,2 2 1 1 1 2 1
++1
```

Result (tab):

```
12 11 12 12 11 12 12 12 13 13 12 13 12 9 7 7
```

Counting the numbers of elements of several short lists is certainly easier than counting the total number. It could also be that large numbers arise when typing if a space is forgotten:

Program 7.7.3: Select larger numbers to check their existence in the lego list.

```
8 4 2 1 15 8 4 1 10 6 6 4
,4 1 1 2 14 4 2 2 4 1 2
,4 2 1 4 4 2 1 2 1 1 1 1
,6 2 1 2 1 2 4 2 2 1 4 1
,4 2 1 2 1 17 11 3 1 2 2
,2 3 4 4 4 11 2 1 2 1 2 4
,7 4 2 4 14 6 5 4 6 4 12 4
,2 2 2 1 6 1 1 2 2 1 1 2
,4 4 2 6 2 2 6 1 4 4 5 5 22
,4 14 4 8 5 8 4 1 6 12 6 2 4
,1 4 2 4 10 6 2 8 1 1 12 1
,1 2 1 1 8 6 6 6 2 1 1 4 4
,2 2 6 2 6 4 17 7 26 2 4 2
,8 12 2 6 6 4 16 8 6
,2 4 3 2 1 2 4
,2 2 1 1 1 2 1
avec ZAHL>9
```

Result (tab)

ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl,	ZAHLl
15 10	14			17 11	11	14 12		22	14 12	10 12		17 26	12 16		

Since each list is of type ZAHLl, the multi-digit numbers are filtered out in each list, which can then be checked again.
Furthermore, you can calculate the sum of each list and discard any that appear to be incorrect.

Program 7.7.4: Calculate the sum for each list.

```
8 4 2 1 15 8 4 1 10 6 6 4
,4 1 1 2 14 4 2 2 4 1 2
,4 2 1 4 4 2 1 2 1 1 1 1
,6 2 1 2 1 2 4 2 2 1 4 1
,4 2 1 2 1 17 11 3 1 2 2
,2 3 4 4 4 11 2 1 2 1 2 4
,7 4 2 4 14 6 5 4 6 4 12 4
,2 2 2 1 6 1 1 2 2 1 1 2
,4 4 2 6 2 2 6 1 4 4 5 5 22
,4 14 4 8 5 8 4 1 6 12 6 2 4
```

```
,1 4 2 4 10 6 2 8 1 1 12 1
,1 2 1 1 8 6 6 6 2 1 1 4 4
,2 2 6 2 6 4 17 7 26 2 4 2
,8 12 2 6 6 4 16 8 6
,2 4 3 2 1 2 4
,2 2 1 1 1 2 1
add ZAHL tup ++
ultimo
```

Result (tab)

```
ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHLl,ZAHL,ZAHLl,ZAHLl
69    37    24    28    46    40    72    23    67    78    52    43    80    68   18    10
```

In the standard representation of a list, the elements are arranged one below the other (vertically). Only single-column lists are often represented horizontally to make better use of the screen. Thus, the above lists are also considered logically vertical, which is why add must be applied and not an assignment.

7.8 Is EXCEL morally worn out?

Program 7.8.1: An o++o program for which EXCEL needs more than six worksheets!

```
<TAB!
NAME,    LENGTH,(AGE,WEIGHT m)m
Klaus    1.68     18   61
                  30   65
                  61   80
Rolf     1.78     40   72
Kathi    1.70     18   55
                  40   70
Walleri  1.00      3   16
Victoria 1.61     13   51
Bert     1.72     18   66
                  30   70
!TAB>
avec NAME! AGE>20
gib BMI,(AGE,BMI,(NAME,BMI m) m) BMI:=WEIGHT:LENGTH:LENGTH!++:
rnd 2
```

If you realize this o++o program in EXCEL you need more than 6 worksheets. Hardly anyone overlooks these EXCEL sheets, which is why they are very difficult to change. More details can be found under o++o versus EXCEL. Spreadsheet programs have several advantages and are widely used, but they also have a number of disadvantages, which we will list:

1. Data and formulas are mixed. For this reason, and because an EXCEL worksheet can contain hundreds or even thousands of formulas, it is almost impossible to check the correctness of the programs or to adapt them to changes.
2. EXCEL does not know schemas for structured tables: e.g. SUBJECT,MARKl l describes a structured schema - here a list of subjects is described, and for each subject there is also a list of marks.
3. EXCEL can display structured tables visually, but it cannot sort them directly or process them reasonably.
4. You cannot use EXCEL to query databases, XML or Wikipedia. For that you would still have to learn SQL, XQuery or better o++o.
5. EXCEL formulas are relatively cryptic because, for example, they often contain individual cell designations. For example, the sum over a column is written in EXCEL in the form:

=SUM(F12:F75)

6. A single EXCEL formula can require more analysis than a complete o++o program.
7. EXCEL contains only a few mathematical concepts and therefore requires an excessive amount of detailed knowledge.
8. EXCEL offers the decimal point to German and other users. However, this makes it difficult to exchange corresponding worksheets of international companies across country borders, since many countries prefer the decimal point.
9. Since data and programs are usually separated in o++o, the data can be used by several programs without any problems. This is more difficult with EXCEL.
10. For aggregations (sums, averages, maxima, ...) per value you have in general to presort or group in EXCEL but not in o++o.
11. In EXCEL, you have to write each number in a separate cell. This could quickly overwhelm a smartphone screen.
12. o++o is based on an abstract tabment concept for data. A tabment can already be represented in many ways by default: web tab xml image column ... and also compact (hsq). With CSS, the output of o++o can be formatted almost arbitrarily. EXCEL, on the other hand, is based on a concrete print image. This makes it easier to create simple applications at first, but it is rather a disadvantage for the complexity of today's applications.
13. After studying the above criticisms of EXCEL, a VW engineer remarked: At VW, EXCEL can be used by any employee at will. As a rule, however, it is only simple tables that are to be made "nice". Sometimes a few simple arithmetic operations are necessary. Comprehensive, complex applications do not take place in EXCEL.

EXCEL does not know mass data operations and could be morally worn out for this reason alone. Therefore I plead for removing EXCEL programs also from school lessons and replacing them by more powerful and promising concepts and systems.

7.9 o++o Proofs

Proofs have played a minor role in school and even outside the world of professional mathematicians. Yet everyone wants to have confidence in a calculation, a system, or a calculator. When confronted with a new type of calculator or system, everyone first tries to solve problems like `2 times 3`. Who suspects further problems, tests e.g.. `1 plus 2 times 3`.
The highly respected German economist Professor Sinn says in his lecture **Energiewende ins Nichts** (see youtube) that calculations only really make sense if you can understand them. To do this, you have to understand all the sub-steps in detail.
We have been working on this requirement of Prof. Sinn for decades. The SQL designers had formulated this requirement somewhat differently at the beginning of their development:
SQL should become an end-user language.
It follows directly that the average consumer should be able to understand SQL programs. Today, however, almost all SQL programmers come from the computer science corner.
The importance of statistics in schools is increasing.
How to teach a student a new statistical function, such as the average `++:` or the function `mad` of o++o. If you simply apply the function to several lists of numbers and look at the result, you usually cannot understand its meaning. However, if the teacher knows that the students have already understood the functions `++` (sum) and `++1` (count), this is no longer so difficult.

Program 7.9.1: Preparation of an o++o proof for the ++: operation..

```
Xl:= 3 5 4 2 1
SUM:=Xl ++
CNT:=Xl ++1
MYAVG:=SUM:CNT
OTTOAVG:=Xl++:
```

Result (ment)

```
<TABM>
 <SUM>15</SUM>
 <COUNT>5</COUNT>
 <AVERAGE>3.</AVERAGE>
 <OTTOAVG>3.</OTTOAVG>
 <X>3</X>
 <X>5</X>
 <X>4</X>
 <X>2</X>
 <X>1</X>
</TABM>
```

Despite this (docu)ment output, it is clear that the two average values match. Here the ment output agrees almost completely with the xml output. The columns MYAVG and OTTOAVG must however agree with all other input lists. The program has the advantage of being very simple. But the student still has to enter a lot of data. Using the example of o++o-mad, which has not yet played a big role in Germany, we want to show that an extended o++o program can relieve us of much of the typing work. This mad function is one of the simplest and clearest statistical functions, but it has not so nice mathematical properties. Now we assume knowledge of the operations `++:`, `..x` and `abs`. By `from ..x to!nr` a list of `nr` random numbers between `from` and `to` is generated. `abs` calculates the absolute value.

Program 7.9.2: o++o Proof for the ++: Operation.

```
RANDOMNRl:= 1 ..x 10!10
Xl:= 1 ..x RANDOMNR!RANDOMNR
AVG:=Xl ++:
DISTANCE:=AVG - X abs
MYMAD:=DISTANCEl ++:
OTTOMAD:=Xl mad
gib AVG,MYMAD,OTTOMAD l
```

Result (tab)

AVG,	MYMAD,	OTTOMAD
2.8	1.44	1.44
4.16666666667	1.5	1.5
1.33333333333	0.444444444444	0.444444444444
1.5	0.5	0.5
5.	2.8	2.8
1.	0.	0.
3.	1.6	1.6
1.	0.	0.
1.	0.	0.
1.	0.	0.

We can easily extend the result table to a thousand output rows table by replacing the last number 10 of the first row with 1000. We have extracted only the relevant columns with the gib statement.

7.10 An example of deep digitization

Perhaps the following example makes the concept of deep digitization (TD) a little clearer: If addition, multiplication, **were not** taught in school, today, for example, you would need different apps to solve the following two problems.

An analogy to deep digitization from the field of "single data" operations

1. I have received a load of 36.57 tons of bulk material and will receive 31 more loads of this type. How much bulk will I have in total?
2. I have a rectangular plot of land 32 m wide and 36.57 m long. What is the size of my plot?

Everyone who has understood multiplication knows that it is one and the same problem that can be solved very easily with a simple calculator. For today's digitization, this means that a TD could require far fewer computer applications than a FD (flat digitization) and that the end users (managers, politicians, ...) could master far more traditional applications (apps). After all, if the apps and applications are based on one (e.g. o++o) data model, one can of course also standardize the interfaces of these apps and such an application could replace many conventional FD applications.

8 Schemes and Structured Tables

All column names of a table are often considered as a schema of the table. Column names are necessary to understand corresponding column values correctly. If we consider structured tables, it is advantageous to enrich the column names with corresponding collection symbols; for example, l for list.

```
NAME,                 BORNIN,       (DEED,                                   YEAR l) l
Otto the Great        Old Saxony(De) Elected King of Germany                  936
                                     The Hungarians defeated on the Lechfeld  955
                                     First emperor of the Holy Roman Empire   962
Otto von Moravia      Moravia        married Euphemia of Hungary             1086
Otto von Guericke     MD(De)         Inventor of the air pump                1649
                                     Hemisphere test for the emperor         1654
Otto von Bismarck     Preussen(De)   with carrot and stick policy            1871
                                     Ems Dispatch                            1870
                                     First Chancellor of Germany             1871
Nicolaus Otto         Taunus (De)    Co-inventor of the gasoline engine      1876
OttoNormalVerbraucher De             learns car driving                      1960
                                     learns a programming language           2025
```

Tabment 8.1: ottos.tab

The above table (TABMENT=TABelle+dokuMENT) ottos.tab contains a list of 6 "persons" and for each person a repeating group (DEED, YEAR l) - a list of (DEED, YEAR) pairs. Here a person has 4 columns, but it is a triple (3-tuple). It is a structured tuple, struple for short (designation of Prof. Schek). The first two components are of type TEXT and the third component is a list of subtuples (pairs) (2-tuples). We will call the attribute values of a level segment. The NAME segment is the same as the BORNIN segment.

The first NAME segment is:

```
NAME,          BORNIN
Otto the Great Old Saxony(De)
```

The first DEED segment of Otto the Great reads:

```
DEED,                   YEAR
Elected King of Germany 936
```

The first person corresponds to the first struple; it is a NAME tuple (=BORNIN tuple):

```
NAME,            BORN,               (DEED,                                   YEAR l)
Otto the Great   Old Saxony(De)       Elected King of Germany                 936
                                      The Hungarians defeated on the Lechfeld 955
                                      First emperor of the Holy Roman Empire  962
```

Since the DEED tuples (= DEED sub-tuples) do not contain any other collections, a DEED segment is the same as a DEED tuple. If we were to represent the above table by an ordinary flat table, every (NAME, BORNIN) pair would have to appear in every row. That is, (Otto the Great, Old Saxony(De)) would have to appear 3 times (once for each DEED segment). Then, for example, it is not so easy to count the persons in the table. With the above table, the corresponding program looks like this:

Program 8.1: How many ottos are contained in the table ? (How many elements (struples) does the outermost collection contain?)

```
ottos.tab
++1
```

Here and in the following, we use these abbreviations and keywords:

aus: from

++1: count

gib: (corresponds to the SELECT of SQL)

avec: with (French) (for selection)

sans: without (for selection)

:= : extension (extends the specified table by a new (complex) column)

m: set: contains different elements

b: Bag: an element may occur more than once

l: list: the order of the elements is important

The result of program 8.1 is a simple table:

ZAHL
6

The schema of this table does not contain a collection symbol because the table contains exactly one element. Similarly, we do not need a collection symbol in the following 2 queries. We do not want to explain the following queries in detail. We use the queries to illustrate what different types of tables there are and what schemas belong to them.

Program 8.2: How many persons and how many deeds are contained in the file ottos.tab?

```
ottos.tab
gib CNTPERSON,CNTDEED
    CNTPERSON:= NAME! ++1
    CNTDEED  := DEED ! ++1
```

Result (tab)

CNTPERSON,	CNTDEED
6	12

Program 8.3: Tell me the name of the person born in Saxony.

```
aus ottos.tab
avec Saxony in BORNIN
gib NAME
```

Result

NAME
Otto the Great

Program 8.4: Give me the name of a noble person.

```
aus  ottos.tab
avec von in NAME
gib  NAME
```

Result

NAME
Otto von Moravia

If keywords like "avec" and "in" are highlighted in color in the future o++o software, the second program line will also be easier to read.

Here it would be better to output the names of all the nobles:

Program 8.5: Sort all noble names

```
aus  ottos.tab
avec von in NAME
gib  NAMEm
```

Result (tab)

```
NAMEl
Otto von Bismarck
Otto von Guericke
Otto von Moravia
```

To save space on the screen or paper, we can also arrange the elements of a list or other collection horizontally:

Result (tabh)

```
NAMEl
"Otto von Bismarck" "Otto von Guericke" "Otto von Moravia"
```

Program 8.6: Count all deeds and all deeds of each century. Add to each century the corresponding people.

```
aus ottos.tab
CENTURY:=YEAR div 100 +1
gib CNTDEED,(CENTURY,CNTDEED,NAMEm m)
    CNTDEED:= DEED ! ++1
```

Result (Table with 3 segment types: CNTDEED, (CENTURY, CNTDEED2) and NAME)

```
CNTDEED,(CENTURY,CNTDEED2,NAMEm m)
12       10      3        Otto the Great
         11      1        Otto von Moravia
         17      2        Otto von Guericke
         19      4        Nicolaus Otto
                          Otto von Bismarck
         20      1        John Doe
         21      1        John Doe
```

Program 8.7: Count all the acts and the acts of each century with corresponding persons, where for each act the corresponding person must appear (with duplicates).

```
aus ottos.tab
CENTURY:=YEAR div 100 +1
gib CNTDEED,(CENTURY,CNTDEED,NAMEb m)
    CNTDEED:= DEED ! ++1
```

Result (tab)

```
CNTDEED,(CENTURY,CNTDEED2, NAMEb m)
12       10      3         Otto the Great
                           Otto the Great
                           Otto the Great
         11      1         Otto von Moravia
         17      2         Otto von Guericke
                           Otto von Guericke
```

19	4	Nicolaus Otto
		Otto von Bismarck
		Otto von Bismarck
		Otto von Bismarck
20	1	John Doe
21	1	John Doe

Where b stands for bag (multiset). So far we have considered only tables with nested levels. But a structured table may also contain "independent" collections:

NAME ,	RESIDENCEl,	WOMANl,	RULESOVERl l
Otto the Great	Magdeburg	Editha	Saxony
	Memleben	Adelheid	Thuringia
			Bavaria
			Franconia
			Swabia
			Italy
			Bohemia
			Holland
			Lorraine
			Friesland
Charles IV	Prague	Margaret	Bohemia
	Tangermünde	Anna	Silesia
		Anna	Brandenburg
		Elizabeth	Italy
			Hungary

Tabment 8.2: emperors.tab

In this table "Memleben" and "Otto the Great" are in the same relation to each other as "Adelheid" and "Otto the Great". But this does not mean that "Adelheid" and "Memleben" are related to each other although they are in the same row. Therefore the following restructuring is senseless.

Program 8.8: Query with empty result

```
aus emperors.tab
gib NAME,RESIDENCE,WIFE m
```

gib NAME,RESIDENCEm,WIFEm m is useful, however.

9 Tabment types (TTs) and structured documents

For structured tables and documents we use the name Tabment. Therefore we abbreviate the type of a tabment with TT (Tabment Type). The TT completes the information given by a schema. It specifies for each tag its schema. For example, the TT for the above table ottos.tab is:

```
TABMENT! OTTOS
OTTOS! NAME,BORNIN,(DEED,YEAR 1)1
NAME BORNIN DEED! TEXT
YEAR! ZAHL
```

TEXT and ZAHL (number) are elementary types that need no further explanation. Each named tabment is surrounded by a tag that is derived from the file name by omitting the type suffix. Therefore, our first table can also be presented in document style or in a document style with inner tables (ment or xml).

for example:

```
<OTTOS>
  <NAME>Otto the Great</NAME>
  <BORNIN>Altsaxony(De)</BORNIN>
  <DEED>Elected  King of Germany</DEED>.
  <YEAR>936</YEAR>
  <DEED>Hungarians beaten on the Lechfeld</DEED>.
  <YEAR>955</YEAR>
  <DEED>First emperor of the Holy Roman Empire</DEED>
  <YEAR>962</YEAR>
  <NAME>Otto von Moravia</NAME>
  <BORNIN>Moravia</BORNIN>
  <DEED>married Euphemia of Hungary</DEED>
  <YEAR>1086</YEAR>
  <NAME>Otto von Guericke</NAME>
  <BORNIN>MD (De)</BORNIN>
  <DEED>Inventor of the air pump</DEED>.
  <YEAR>1649</YEAR>
  <DEED>Half ball attempt in front of the emperor</DEED>.
  <YEAR>1654</YEAR>
  <NAME>Otto von Bismarck</NAME>
  <BORNIN>Preussen(De)</BORNIN>
  <AT>with carrot and stick policy</DEED>.
  <YEAR>1871</YEAR>
  <DEED>Ems Dispatch</DEED>.
  <YEAR>1870</YEAR>
  <AT>First Chancellor of the Reich of Germany</DEED>.
  <YEAR>1871</YEAR>
  <NAME>Nicolaus Otto</NAME>
  <BORNIN>Taunus (De)</BORNIN>
  <DEED>Miter inventor of the gasoline engine</DEED>.
  <YEAR>1876</YEAR>
  <NAME>Otto Normal Consumer</NAME>
  <BORNIN>De</BORNIN>
  <DEED>learns to drive</DEED>
  <YEAR>1960</YEAR>
  <DEED>learns a programming language</DEED>.
  <YEAR>2025</YEAR>
</OTTOS>
```

Tabment 9.1: Table ottos.tab in XML document style

```
"Otto the Great" Old Saxony(De)
```

```
 "Elected King of Germany" 936
 "The Hungarians defeated on the Lechfeld" 955
 "First Emperor of the Holy Roman Empire" 962
"Otto of Moravia" Moravia
 "married Euphemia of Hungary" 1086
"Otto von Guericke" "MD (De)"
 "Inventor of the air pump" 1649
 "Hemisphere trial before the emperor" 1654
"Otto von Bismarck" Prussia(De)
 "with carrot and stick policy" 1871
 "Ems Dispatch" 1870
 "First Imperial Chancellor of Germany" 1871
"Nicolaus Otto" "Taunus (De)"
 "Co-inventor of the gasoline engine" 1876
"Otto Normalverbraucher" De
 "learns to drive" 1960
 "learns a programming language" 2025
```

Tabment 9.2: Table ottos.tab in hsq style

Let's look at parts of a small but real document: "Basic Law for the Federal Republic of Germany".

```
<META!
TABMENT ! BASICLAW
BASICLAW ! NR,TITLE,CONTENT 1
NR ! ONR
TITLE ! TEXT
CONTENT ! TEXT
!META>
<BASICLAW>
  <NR>1</NR>
  <TITLE>The Fundamental Rights</TITLE>
  <CONTENT></CONTENT>
  <NR>1.1</NR>
  <TITLE>Human Dignity - Human Rights - Binding of Fundamental Rights</TITLE>
  <CONTENT></CONTENT>
  <NR>1.1.1</NR>
  <TITLE></TITLE>
  <CONTENT>The dignity of the human being is inviolable. To respect and protect it is the obligation
of all state power.</CONTENT>
  <NR>1.1.2</NR>
  <TITLE></TITLE>
  <CONTENT>The German people therefore profess inviolable and inalienable human rights as the basis
of every human community, peace and justice in the world.</CONTENT>
  <NR>1.1.3</NR>
  <TITLE></TITLE>
  <CONTENT>The following fundamental rights bind legislation, executive power and jurisdiction as
directly applicable law.</CONTENT>
  <NR>1.2</NR>
  <TITLE>Personal Freedom</TITLE>
  <CONTENT></CONTENT>
  <NR>1.2.1</NR>
  <TITLE></TITLE>
  <CONTENT>Everyone has the right to the free development of his personality, insofar as he is
        does not infringe the rights of others and does not violate the constitutional order or the
        Moral law violates.</CONTENT>
  <NR>1.2.2</NR>
  <TITLE></TITLE>
  <CONTENT>Everyone has the right to life and ...</CONTENT>
  <NR>1.14</NR>
  <TITLE>Property Inheritance Expropriation</TITLE>
  <CONTENT></CONTENT>
  <NR>1.14.1</NR>
  <TITLE></TITLE>
  <CONTENT>The property and inheritance rights are guaranteed. The content and limits are
         determined by the laws.</CONTENT>
  <NR>1.14.2</NR>
  <TITLE></TITLE>
  <CONTENT>Ownership obliges. Its use shall at the same time be for the public good.
```

```
          serve.</CONTENT>
  <NR>1.14.3</NR>
  <TITLE></TITLE>
  <CONTENT>Expropriation is permissible only for the public good. It may only be carried out by
         law or on the basis of a law regulating the type and extent of compensation.
         Compensation shall be paid after fair consideration of the interests of the general public
and the
         to be determined by the parties involved. In the event of a dispute, the amount of
compensation shall be determined by the
         Legal recourse before the ordinary courts open.</CONTENT>
  <NR>1.19</NR>
  <TITLE>Restriction of Fundamental Rights</TITLE>
  <CONTENT></CONTENT>
  <NR>1.19.1</NR>
  <TITLE></TITLE>
  <CONTENT>To the extent that_under_this_foundation_law a fundamental right is protected by law
...</CONTENT>
  <NR>2</NR>
  <TITLE>The Federal Government and the States</TITLE>
  <CONTENT></CONTENT>
  <NR>2.1</NR>
  <TITLE>Fundamentals of State Order, Right of Resistance</TITLE>
  <CONTENT></CONTENT>
  <NR>2.1.1</NR>
  <TITLE></TITLE>
  <CONTENT>The_Federal_Republic_of_Germany is a democratic and social federal state.</CONTENT>
  <NR>2.1.2</NR>
  <TITLE></TITLE>
  <CONTENT>All state power emanates from the people. It is exercised by the people in elections and
votes ...</CONTENT>
</BASICLAW>
```

Tabment 9.3: basiclaw1.ment

```
<META!
TABMENT! BASICLAW
BASICLAW! CHAPTERl
CHAPTER! KNR,KTITLE,ARTICLEl
ARTICLE! ANR,ATITLE,(PNR,PARAGRAPH l)
KNR ANR PNR! ZAHL
KTITLE ATITLE PARAGRAPH! TEXT
!META>
<BASICLAW>
  <CHAPTER>
    <KNR>1</KNR>
    <KTITLE>The Fundamental Rights</KTITLE>
    <ARTICLE>
      <ANR>1</ANR>
      <ATITLE>Human Dignity - Human Rights - Binding of Fundamental Rights</ATITLE>
      <PNR>1</PNR>
      <PARAGRAPH>The dignity of the human being is inviolable. To respect and protect it is the duty
of all_state_authority.</PARAGRAPH>
      <PNR>2</PNR>
      <PARAGRAPH>The German people therefore profess inviolable and inalienable human rights as the
basis of every human community, of peace and justice in the world.</PARAGRAPH>
      <PNR>3</PNR>
      <PARAGRAPH>The following fundamental rights bind legislation, executive power and jurisdiction
as directly applicable law.</PARAGRAPH>
    </ARTICLE>
    <ARTICLE>
      <ANR>2</ANR>
      <ATITLE>Personal Freedom</ATITLE>
      <PNR>1</PNR>
      <PARAGRAPH>Everyone has the right to the free development of his or her personality, insofar
as he or she is
        does not violate the rights of others and does not violate the constitutional_order or the
        Moral Law Violates.</PARAGRAPH>
      <PNR>2</PNR>
      <PARAGRAPH>Everyone has the right to life and ...</PARAGRAPH>
    </ARTICLE>
    <ARTICLE>
      <ANR>14</ANR>
      <ATITLE>Property Inheritance Expropriation</ATITLE>
      <PNR>1</PNR>
      <PARAGRAPH>The property and inheritance rights are guaranteed. The content and limits are
         determined by the laws.</PARAGRAPH>
```

```
      <PNR>2</PNR>
      <PARAGRAPH>Ownership obliges. Its use shall at the same time be for the public good.
          serve.</PARAGRAPH>
      <PNR>3</PNR>
      <PARAGRAPH>Eminent domain is permissible only for the public good. It may only be carried out 
by
         law or on the basis of a law regulating the type and extent of compensation.
         Compensation shall be paid after fair_consideration_of the interests of the general public 
and the
         to be determined by the parties involved. In the event of a dispute, the amount of 
compensation shall be determined by the
         legal courts open to the ordinary courts.</PARAGRAPH>
    </ARTICLE>
    <ARTICLE>
      <ANR>19</ANR>
      <ATITLE>Restriction of Fundamental Rights</ATITLE>
      <PNR>1</PNR>
      <PARAGRAPH>So far as under this foundation law a fundamental right is established by law 
...</PARAGRAPH>
    </ARTICLE>
  </CHAPTER>
  <CHAPTER>
    <KNR>2</KNR>
    <KTITLE>The Federal Government and the States</KTITLE>
    <ARTICLE>
      <ANR>20</ANR>
      <ATITLE>Fundamentals of State Order, Right of Resistance</ATITLE>
      <PNR>1</PNR>
      <PARAGRAPH>The Federal Republic of Germany is a democratic and social_Federal 
State.</PARAGRAPH>
      <PNR>2</PNR>
      <PARAGRAPH>All state power emanates from the people. It is exercised by the people in 
elections_and_votes ...</PARAGRAPH>
    </ARTICLE>
  </CHAPTER>
</BASICLAW>
```

Tabment 9.4 basiclaw2.ment

basiclaw1.ment and basiclaw2.ment contain the same information, yet differ significantly in the way they are structured. The metadata of the second version is based on the designations of the original basic law. Here a structuring depth of 3 is present by the terms CHAPTER, ARTICLE and PARAGRAPH, which could be increased actually only by new names. The metadata from the first document, on the other hand, can be used for any structured documents. Although there are only 3 "column names" (NR, TITLE, CONTENT), the depth of "structuring" is unlimited.

Documents containing the "new line" character (return) are only conditionally suitable for tab and hsq outputs. Structured documents are usually output as ment or xml. For (NR,TITLE,CONTENT l) documents even the tabular web output is suitable. That the web output is advantageous for the comparison of both document types above, shall be demonstrated now.

Program 9.1: Output a document in the form of two different tables.
`basiclaw1.ment,basiclaw2.ment`
Result (web)

BASICLAW1

NR	TITLE	CONTENT
1	The Fundamental Rights	
1.1	Human Dignity - Human Rights - Binding of Fundamental Rights	
1.1.1		The dignity of the human being is inviolable. To respect and protect it is the obligation of all state power.
1.1.2		The German people therefore profess inviolable and inalienable human rights as the basis of every human community, peace and justice in the world.
1.1.3		The following fundamental rights bind legislation, executive power and jurisdiction as directly applicable law.
1.2	Personal Freedom	
1.2.1		Everyone has the right to the free development of his personality, insofar as he is does not infringe the rights of others and does not violate the constitutional order or the Moral law violates.
1.2.2		Everyone has the right to life and ...
1.14	Property Inheritance Expropriation	
1.14.1		The property and inheritance rights are guaranteed. The content and limits are determined by the laws.
1.14.2		Ownership obliges. Its use shall at the same time be for the public good. serve.
1.14.3		Expropriation is permissible only for the public good. It may only be carried out by law or on the basis of a law regulating the type and extent of compensation. Compensation shall be paid after fair consideration of the interests of the general public and the to be determined by the parties involved. In the event of a dispute, the amount of compensation shall be determined by the Legal recourse before the ordinary courts open.
1.19	Restriction of Fundamental Rights	
1.19.1		To the extent that under this_foundation_law a fundamental right is protected by law ...
2	The Federal Government and the States	
2.1	Fundamentals of State Order, Right of Resistance	
2.1.1		The_Federal_Republic_of Germany is a democratic and social federal state.
2.1.2		All state power emanates from the people. It is exercised by the people in elections and votes ...

BASICLAW2

KNR	KTITLE	ANR	ATITLE	PNR	PARAGRAPH
1	The Fundamental Rights	1	Human Dignity - Human Rights - Binding of Fundamental Rights	1	The dignity of the human being is inviolable. To respect and protect it is the duty of all_state_authority.
				2	The German people therefore profess inviolable and inalienable human rights as the basis of every human community, of peace and justice in the world.
				3	The following fundamental rights bind legislation, executive power and jurisdiction as directly applicable law.
		2	Personal Freedom	1	Everyone has the right to the free development of his or her personality, insofar as he or she is does not violate the rights of others and does not violate the constitutional_order or the Moral Law Violates.
				2	Everyone has the right to life and ...
		14	Property Inheritance Expropriation	1	The property and inheritance rights are guaranteed. The content and limits are determined by the laws.
				2	Ownership obliges. Its use shall at the same time be for the public good. serve.
				3	Eminent domain is permissible only for the public good. It may only be carried out by law or on the basis of a law regulating the type and extent of compensation. Compensation shall be paid after fair_consideration_of the interests of the general public and the to be determined by the parties involved. In the event of a dispute, the amount of compensation shall be determined by the legal courts open to the ordinary courts.
		19	Restriction of Fundamental Rights	1	So far as under this foundation law a fundamental right is established by law ...
2	The Federal Government and the States	20	Fundamentals of State Order, Right of Resistance	1	The Federal Republic of Germany is a democratic and social_Federal State.
				2	All state power emanates from the people. It is exercised by the people in elections_and_votes ...

Program 9.2: Calculate for each chapter the number of paragraphs twice.

```
aus basiclaw1.ment,basiclaw2.ment
KNR1:=NR nthzahl 1 at NR
PNR1:=NR nthzahl 3 leftat TITLE
gib KNR1,PCNT1 m,(KNR,PCNT m)
    PCNT1:=PNR1!++1
    PCNT :=PNR !++1
```

Result (tab)

```
KNR1,PCNT1 l,(KNR,PCNT l)
```

```
   1     9      1    9
   2     2      2    2
```

Program 9.3: Calculate twice the number of characters (letters) of each article.

```
basiclaw1.ment,basiclaw2.ment
ANR1:=NR nthzahl 1 text + "." + (NR nthzahl 2 text) onr
gib ANR1,CNT1 m, (ANR,CNT2 m)
    CNT1 :=CONTENT zil ++1! ++
    CNT2 :=PARAGRAPH zil ++1! ++ # zil creates list of Zi (characters)
```

Result (tab)

```
ANR1, CNT1 l, (ANR, CNT2 l)
```

```
1.1   366       1   367
1.2   250       2   263
1.14  672      14   667
1.19   88      19    81
2.1   171      20   171
```

Program 9.4: Calculate the number of characters and words of each article.

```
basiclaw1.ment
ANR1:=NR nthzahl 1 text + "." + (NR nthzahl 2 text) onr
gib ANR1,CNT1,CNT2 m
    CNT1 :=CONTENT zil ++1! ++
    CNT2 :=CONTENT cil ++1! ++ # cil generates list of ci (words)
```

Result (tab)

```
ANR1, CNT1, CNT2 l
```

```
1.1   366    57
1.2   250    41
1.14  672   106
1.19   88    13
2.1   171    26
```

Zi and Ci are Chinese words meaning character and word, respectively.

The example makes it clear that a teacher can easily count through the essays of a class in this way. We can further see that queries to documents are not very different from queries to tables, since documents can also be thought of as tables.

The functions described here could be useful for typical tasks of editorial offices or translation agencies, among others.

Program 9.5: Give me Article 2 of the Basic Law.

```
aus  basiclaw1.ment
avec NR subtext 1!3="1.2"
gib  TITLE,CONTENT l
```

Result (ment):

```
TABMENT ! TABM
TABM ! TITLE,CONTENT l
TITLE ! TEXT
CONTENT ! TEXT
```

```
<TABM>
  <TITLE>Personal Freedom</TITLE>
  <CONTENT></CONTENT>
  <TITLE></TITLE>
  <CONTENT>Everyone has the right to the free development of his personality, insofar as he is
        does not infringe the rights of others and does not violate the constitutional order or the
        Moral law violates.</CONTENT>
  <TITLE></TITLE>
  <CONTENT>Everyone has the right to life and ...</CONTENT>
</TABM>
```

Program 9.6: Find an entire section with a specific word in the title.

```
aus  basiclaw1.ment
avec expropriation in TITLE
gib  NR
rename NR! NR0
L:=NR0 zil ++1
,basiclaw1.ment
avec NR subtext 1!L = NR0
gib NR,TITLE,CONTENT l
```

Result (ment):

```
TABMENT ! TABM
TABM ! NR,TITLE,CONTENT l
NR ! ONR
TITLE ! TEXT
CONTENT ! TEXT
```

```
<TABM>
  <NR>1.14</NR>
  <TITLE>Property Inheritance Expropriation</TITLE>
  <CONTENT></CONTENT>
  <NR>1.14.1</NR>
  <TITLE></TITLE>
  <CONTENT>The property and inheritance rights are guaranteed. The content and limits are
         determined by the laws.</CONTENT>
  <NR>1.14.2</NR>
  <TITLE></TITLE>
  <CONTENT>Ownership obliges. Its use shall at the same time be for the public good.
          serve.</CONTENT>
  <NR>1.14.3</NR>
  <TITLE></TITLE>
  <CONTENT>Expropriation is permissible only for the public good. It may only be carried out by
         law or on the basis of a law regulating the type and extent of compensation.
         Compensation shall be paid after fair consideration of the interests of the general public
and the
         to be determined by the parties involved. In the event of a dispute, the amount of
compensation shall be determined by the
         Legal recourse before the ordinary courts open.</CONTENT>
</TABM>
```

We consider another document with TT. It uses alternatives through (|). It comes from the XQuery use cases (C+07).

```
<META!
TABMENT! REPORT1
REPORT1! SECTIONl
SECTION! TITLE,CONTENT
CONTENT! TEXT|NARCOSIS|PREPARATION|CUT|ACTION|OBSERVATION l
PREPARATION! TEXT|ACTION l
CUT! TEXT|GEOGRAPHY|INSTRUMENT l
ACTION! TEXT|INSTRUMENT l
TITLE NARCOSIS OBSERVATION GEOGRAPHY INSTRUMENT! TEXT
!META>
<REPORT1>
  <SECTION>
    <TITLE>Procedure</TITLE>
    <CONTENT>
  The patient was taken to the operating room, where she was placed in the supine position and
<NARCOSIS> induced under general anesthesia. </NARCOSIS>
<PREPARATION>
<ACTION>A Foley catheter was placed to decompress the bladder</ACTION> and the abdomen was then
sterilely prepped and draped.
</PREPARATION>
<CUT>
A curved incision was made
<GEOGRAPHY> in the center line immediately infraumbilical </GEOGRAPHY>
 and the subcutaneous tissue was divided
 <INSTRUMENT> Use electrocautery. </INSTRUMENT>
 </CUT>
 The fascia was identified and
 <ACTION> # 2 0 Maxon seams were placed on each side of the centerline.
 </ACTION>
 <CUT>
 The fascia was shared with
<INSTRUMENT> electrocautery </INSTRUMENT>
 and the peritoneum entered.
</CUT>
 <OBSERVATION>The small intestine was identified.</OBSERVATION>
 and
 <ACTION> the <INSTRUMENT>Hasson trocar</INSTRUMENT>
 was placed under direct visualization.
 </ACTION>
 <ACTION>The <INSTRUMENT>Trocar</INSTRUMENT>using the
      Sutures was attached to the fascia.
</ACTION>
</CONTENT>
</SECTION>
</REPORT1>
```

Tabment 9.5: report1.ment

In report1.xml the CONTENT is a list of elements, where each element is either of type TEXT, ANESTESIA, PREPARATION, CUT, ACTION or OBSERVATION. In the above document, the first element is simple TEXT, the second is of type ANESTESIA, the third is of type PREPARATION, ... Since our report was tagged in the above way, the following example queries are possible.

For example:

Program 9.6: What instruments were used in the second cut?

```
aus  report1.ment
gib  CUTl
avec CUT pos = 2
gib  INSTRUMENTl
```

Result (tab)

INSTRUMENTl
electrocautery

Program 9.7: What are the first two instruments used?

```
aus  report1.ment
gib  INSTRUMENTl
```

avec INSTRUMENT pos < 3
Result (tab)
INSTRUMENT1
Use electrocautery. electrocautery

10 A university database

We consider a non-relational database consisting of one flat and two structured tables:

```
FACS! FAC,DEAN,BUDGET,STUDCAPACITY m
STUDENTS! STID,NAME,LOCATION?,STIP,FAC,(COURSE,MARK m),(PROJ,HOURS m) m
COURSES! COURSE,TEACHER,(ISBN,TITLE m)m
```

The underlined column names are keys. The last two tables can be represented by the following 5 flat relations:

```
student1: STID,NAME,LOCATION?,STIP,FAC m
exam1: STID,COURSE,MARK m
projects1: STID,PROJ,HOURS m
course1: COURSE,TEACHER m
course_books1: COURSE,ISBN,TITLE m
```

FAC,	DEAN,	BUDGET,	STUDCAPACITY m
Art	Sitte	2'000	600
Infor	Reichel	10'000	500
Math	Dassow	1'000	200
Philo	Hegel	1'000	10
Sport	Streich	8'000	150

Tabment 10.1: facs.tab

STID,	NAME,	LOC?,	STIP,	FAC,	(COURSE,	MARK m),	(PROJ,	HOURS m)m
1111	Ernst	Oehna	500	Math	Algebra	1	Fritz	4
					Logic	2	Otto	2
					History	1		
2222	Sophia	Berlin	400	Infor	Algebra	3	Ghandi	5
					Databases	1	Ming	4
					Otto	1	Otto	6
3333	Clara	Oehna	450	Infor	Databases	1		
					OCaml	2		
4444	Ulrike		400	Art			Monet	10
5555	Käthe	Gerwisch	600	Art	Repin	1	Monet	20
					Apel	1		
6666	Claudia	Berlin	600	Sport	Psycho	2	Matthes	8
					Ski	1	Witt	12

Tabment 10.2: students.tab

COURSE,	TEACHER,	(ISBN,	TITLE m)m
Algebra	Reichel	0138-3019	Structural Induction on Partial Alg.
		3-8244-2099-6	Structured tables
Databases	Saake	0-321-31256-2	Database Systems an Application
		0-7167-8069-0	Principles of Database Systems
Otto	Benecke	0-7167-8069-0	Principles of Database Systems
		3-8244-2099-6	Structured tables

Tabment 10.3: courses.tab

STID,	NAME,	LOC?,	STIP,	FAC m
1111	Ernst	Oehna	500	Math
2222	Sophia	Berlin	400	Infor
3333	Clara	Oehna	450	Infor
4444	Ulrike		400	Art
5555	Käthe	Gerwisch	600	Art
6666	Claudia	Berlin	600	Sport

Tabment 10.4: students1.tab

```
STID,COURSE, MARK m
1111 Algebra   1
1111 History   1
1111 Logic     2
2222 Algebra   3
2222 Databases 1
2222 Otto      1
3333 Databases 1
3333 OCaml     2
5555 Apel      1
5555 Repin     1
6666 Psycho    2
6666 Ski       1
```

Tabment 10.5: examen1.tab

```
STID,PROJ,  HOURS m
1111 Fritz    4
1111 Otto     2
2222 Ghandi   5
2222 Ming     4
2222 Otto     6
4444 Monet   10
5555 Monet   20
6666 Matthes  8
6666 Witt    12
```

Tabment 10.6: projects1.tab

The above tables and the following programs refer to tab files, although we keep in mind that the specified tables could be database tables.

10.1 Selection (avec sans)

A condition specifies tuples or subtuples. In an avec clause the specified tuples form the result, in a sans clause the specified tuples are omitted.

Consequently, the schema and the TT of the considered tabment are not changed by a selection. Column names or tags are written in upper case in an o++o program. They must start with a letter or the character "_". A WORT (word) that is not enclosed by "-symbols must therefore use a lowercase letter. TEXT may contain spaces; however, they must then be enclosed in "-symbols.

Program 10.1.1: Find all students from Berlin and Oehna with bad results..

```
aus  students.tab
avec LOC in "Berlin Oehna" # selected students
avec MARK > 2 # selects exams and students
```

Result (tab)

```
STID,NAME , LOC?  ,STIP,FAC ,(COURSE ,MARK m),  (PROJ , HOURS m) m
2222 Sophia Berlin 400  Infor Algebra 3          Ghandi  5
                                                 Ming    4
                                                 Otto    6
```

Intermediate result after the first condition

```
1111 Ernst  Oehna  500 Math      Algebra 1       Fritz  4
                                 History 1       Otto   2
                                 Logic   2
2222 Sophia Berlin 400 Infor     Algebra 3       Ghandi 5
```

```
                                Databases 1     Ming     4
                                Otto      1     Otto     6
3333 Clara   Oehna  450 Infor   Databases 1
                                OCaml     2
6666 Claudia Berlin 600 Sport   Psycho    2     Matthes  8
                                Ski       1     Witt    12
```

The second "condition" is applied to the result of the first condition. The second "condition" is an abbreviation for the following two conditions:

```
avec STID!   MARK>2 # Selection STID tuple (MARK>2 must exist)
avec COURSE! MARK>2 # Selection COURSE tuple
```

The first of these two conditions expresses that we select (complete) student tuples for which there exists a (COURSE,MARK) subtuple with a grade of 3 or higher. We do not write the existence quantifier because there is exactly one EXIST quantifier behind each condition. "#" is the comment symbol. It can be used to describe the meaning of a program step. Also, lines can be commented out to indicate intermediate results.

Program 10.1.2: For all students from Oehna and Berlin, indicate all results with 3 or worse.

```
aus   students.tab
avec  LOC in "Berlin Oehna" # equivalent: LOC in Berlin Oehna
avec  COURSE! MARK>2        # selects exams and not students
gib   NAME,LOC,(COURSE,MARK m) b
```

Result (tab)

```
NAME ,  LOC , (COURSE, MARK m) b
Clara   Oehna
Claudia Berlin
Ernst   Oehna
Sophia  Berlin Algebra 3
```

After applying the two conditions, the restructuring (see section 10.3) was applied. Therefore, the scheme of the result has changed and the data has been sorted.

Program 10.1.3: Find all students from Oehna and Berlin with a grade of 3 or worse, with all scores.

```
aus  students.tab
avec LOC in "Berlin Oehna"
avec STID! MARK>2 # selects only students and not exams
gib  NAME,LOC,(COURSE,MARK m)b
```

Result (tab)

```
NAME , LOC ,  (COURSE,   MARK m) b
Sophia Berlin  Algebra    3
               Databases 1
               Otto      1
```

Program 10.1.4: Find all students who have only a grade of 1 and at least one grade of 1..

```
aus students.tab
avec NOTEm = {1} # { } are set brackets
```

Result (tab)

```
STID,NAME , LOC?    ,STIP,FAC ,(COURSE,MARK m),(PROJ ,HOURS m) m
5555 Käthe  Gerwisch 600  Art   Apel    1         Monet 20
                                Repin   1
```

For the evaluation of the condition, for each student the list of his grades is transformed into a set. Thus, Ernst's set {1 2 1} = {1 2} and Kathe's set {1 1} is equal to {1}. Two sets are equal if every element of the left side is also on the right side and every element of the right side is on the left side. In other words, two sets M1 and M2 are equal if 'M1 inmath M2 & M2 inmath M1' holds. If we want to have all students with exactly two marks 1, then we can use multisets: MARKb = {{1 1}} (b abbreviates Bag). If the order of the notes is also important, then we can take lists: MARKl = [1 2 1], ...

Program 10.1.5: Find all students who got an 1 in the algebra course..

```
aus  students.tab
avec STID! COURSE=Algebra & MARK=1
gib  STID,NAME,(COURSE,MARK m)m
```

Result (tab)

```
STID,NAME ,(COURSE, MARK m) m
1111 Ernst  Algebra 1
            History 1
            Logic   2
```

Program 10.1.6: Find all students who have taken an algebra course and have an 1 (not necessarily in the same course)..

```
aus  students.tab
avec STID! COURSE=Algebra
avec STID! MARK=1
gib  STID,NAME,(COURSE,MARK m)m
```

Result (tab)

```
STID,NAME , (COURSE ,  MARK m) m
1111 Ernst   Algebra   1
             History   1
             Logic     2
2222 Sophia  Algebra   3
             Databases 1
             Otto      1
```

Program 10.1.7: Find all students who already have exams in Algebra and Databases..

```
aus  students.tab
avec STID! COURSE=Algebra
avec STID! COURSE=Databases
# avec Algebra Databases in COURSEm is equivalent to both selections
```

Result (tab)

```
STID,NAME , LOC?  ,STIP,FAC ,  (COURSE ,  MARK m),(PROJ , HOURS m) m
2222 Sophia Berlin 400  Infor   Algebra   3         Ghandi 5
                                Databases 1         Ming   4
                                Otto      1         Otto   6
```

Intermediate result after the first condition

```
1111 Ernst Oehna   500  Math    Algebra   1         Fritz  4
```

```
                                History   1        Otto   2
                                Logic     2
2222 Sophia Berlin 400  Infor   Algebra   3        Ghandi 5
                                Databases 1        Ming   4
                                Otto      1        Otto   6
```

If we would connect both conditions by & (and), this condition "contains" only one EXIST quantifier, of the kind that no subtuple exists that satisfies both subconditions simultaneously. The result would be empty in any case.

Program 10.1.8: For each student who has completed Algebra, indicate all other courses they have completed.

```
aus  students.tab
avec STID!   COURSE=Algebra # selects students
sans COURSE! COURSE=Algebra # chooses exam
gib  NAME,COURSEb m
```

Result (tabh)

```
NAME,   COURSEb m
Ernst   History Logic
Sophia  Databases Otto
```

Program 10.1.9: Find all students in which the word Otto occurs

```
aus  students.tab
avec Otto
```

Result (tab)

```
STID,NAME , LOC?  ,STIP,FAC ,  (COURSE ,MARK m),(PROJ , HOURS m) m
1111 Ernst  Oehna  500  Math    Algebra   1       Fritz  4
                                History   1       Otto   2
                                Logic     2
2222 Sophia Berlin 400  Infor   Algebra   3       Ghandi 5
                                Databases 1       Ming   4
                                Otto      1       Otto   6
```

Program 10.1.10: Print from all tuples of the university database (to which I have access) the tuples containing the word Apel.

```
aus  students.tab,courses.tab
avec Apel
```

Result (xml)

```
<TABM>
  <STUDENTS>
    <STID>5555</STID>
    <NAME>Käthe</NAME>
    <Place>Gerwisch</Place>
    <STIP>600</STIP>
    <FAC>Art</FAC>
    <COURSE>Apel</COURSE>.
    <MARK>1</MARK>
    <COURSE>Repin</COURSE>
    <MARK>1</MARK>
```

```
    <PROJ>Monet</PROJ>
    <HOURS>20</HOURS>
  </STUDENTS>
  <COURSES/>
</TABM>
```

So far in this section we have only considered "selection by content", but almost the same importance has "selection by position". This is not only useful for lists, but can also be used in the context of "relational applications". We only consider two examples here.

Program 10.1.11: Give for each student from Oehna with exams, the last exam.

```
aus  students.tab
avec LOC=Oehna
avec MARK pos- = 1
gib  STID,NAME,(COURSE,MARK m)m
```

Result (tab)

```
STID,NAME,(COURSE,MARK m) m
1111 Ernst Logic  2
3333 Clara OCaml  2
```

The pos (pos-) function returns the position number (position number backwards) of the (sub-) item in the corresponding set. Therefore, MARK pos is the same as COURSE pos.

Program 10.1.12: Give the 2 best exams for the 3 best students. We omit Ulrike because we cannot calculate an average for her. She has no grades yet.

```
aus students.tab
sans NAME=Ulrike
avec MARK=MARK
gib  AVGM,NAME,FAC,(MARK,COURSE m)m
     AVGM:= MARK! ++:
avec NAME pos < 4
avec MARK pos < 3
rnd 2
```

Result (tab)

```
AVGM,NAME ,FAC ,  (MARK,COURSE m ) m
1.00 Käthe Art     1    Apel
                   1    Repin
1.33 Ernst Math    1    Algebra
                   1    History
1.50 Clara Infor   1    Databases
                   2    OCaml
```

Here it is sufficient to know that by the gib clause the students are sorted by AVGM, the exams are sorted by MARK and AVGM is the average for each student. The gib clause is explained in more detail in section 10.3.

Although the following query does not require *avec* or *sans*, *primo* and *ultimo* are still selections. *ultimo* selects the last element from each collection. These operations can be used to quickly get a first impression of the structure and content of a tab.

Program 10.1.13: Find the last element of each collection of the student file.

```
students.tab ultimo
```

Result (tab)

STID ,NAME	,LOC?	,STIP	,FAC	,(COURSE	,MARK	m),(PROJ	,HOURS	m) m
6666 Claudia	Berlin	600	Sport	Ski	1	Witt	12	

Program 10.1.14: Find the students with the highest scholarships.

```
aus students.tab
STIPMAX:=STIPl max # a new column with a value is created
avec STIP=STIPMAX
```

Result (tab)

STIPMAX	,(STID	,NAME	,LOC?	,STIP	,FAC	,(COURSE	,MARK	m),(PROJ	,HOURS	m) m)
600	5555	Käthe	Gerwisch	600	Art	Apel	1	Monet	20	
						Repin	1			
	6666	Claudia	Berlin	600	Sport	Psycho	2	Matthes	8	
						Ski	1	Witt	12	

10.2 Calculations (:=)

Program 10.2.1: Calculate the gross values of several prices.

```
3.18 55.88 17.90 * 1.19
```

Result (hsqh and tabh)

PZAHLl
3.7842 66.4972 21.301

Program 10.2.2: Calculate the gross values of several prices and leave the entered values in the output.

```
NETl:=3.18 55.88 17.90
GROSS:=NET*1.19
```

Result (tab)

NET,	GROSS l
3.18	3.7842
55.88	66.4972
17.9	21.301

Program 10.2.3: Calculate the gross values of several prices.

```
3.18 55.88 17.90 +% 19
```

Result (tabh)

PZAHLl
3.7842 66.4972 21.301

Program 10.2.4: Convert all net prices of a small table into gross prices..

```
<TAB!
ARTICLE,  PRICE l
OttoRAMDB 500
OttoWiki   10
OttoCalc   20
!TAB>
+% 19
```

Result

```
ARTICLE , PRICE l
OttoRAMDB 595.
OttoWiki   11.9
OttoCalc   23.8
```

19% is added to each value in the table. Text values are not changed by arithmetic operations with numbers.

Program 10.2.5: Calculate the gross value of each item and the sum of all gross values.

```
<TAB!
ARTICLE,  PRICE,CNT m
OttoRAMDB 500     20
OttoWiki   10    200
OttoCalc   20   4000
!TAB>
TOTAL:=PRICE*CNT +% 19
TOTALSUM:=TOTALl ++
```

Result

```
TOTALSUM,(ARTICLE ,PRICE,CNT, TOTAL m)
 109480.  OttoCalc   20  4000 95200.
          OttoRAMDB 500    20 11900.
          OttoWiki   10   200 2380.
```

Program 10.2.6: Report each computer science student's stipend in dollars..

```
aus  students.tab
avec FAC=Infor
DOL:=STIP*1.02
```

Result

```
STID,NAME , LOC?  ,STIP,FAC , DOL , (COURSE   , MARK m),(PROJ , HOURS m) m
2222 Sophia Berlin 400  Infor 408.   Algebra    3        Ghandi 5
                                     Databases  1        Ming   4
                                     Otto       1        Otto   6
3333 Clara  Oehna  450  Infor 459.   Databases  1
                                     OCaml      2
```

Program 10.2.7: Pay each student 100 euros for each of their projects..

```
aus students.tab
BONUS:= PROJl ++1 *100
gib STID,NAME,BONUS m
```

Result

```
STID,NAME,   BONUS m
1111 Ernst   200
2222 Sophia  300
3333 Clara     0
4444 Ulrike  100
5555 Käthe   100
6666 Claudia 200
```

Program 10.2.8: Pay each Oehna student an additional bonus based on their grade point average..

```
aus students.tab
avec LOC=Oehna
AVG1:= MARKl ++:
BONUS3:=1000 : AVG1
```

```
gib STID,NAME,AVG1,BONUS3 m
rnd 2
```

Result

```
STID,NAME ,AVG1,BONUS3 m
1111 Ernst 1.33 750.00
3333 Clara 1.50 666.67
```

With the help of rnd (round) every value of a table is rounded to 2 digits after point (dot). For texts the value remains unchanged again.

Program 10.2.9: The students of the math faculty get a bonus of 900 euros, the computer science of 800 euros and all others get 700 euros..

```
aus students.tab
BONUS:= 900 if FAC=Math !
        800 if FAC=Infor!
        700
gib STID,NAME,FAC, BONUS m
```

Result

```
STID, NAME ,  FAC , BONUS m
1111  Ernst   Math  900
2222  Sophia  Infor 800
3333  Clara   Infor 800
4444  Ulrike  Art   700
5555  Käthe   Art   700
6666  Claudia Sport 700
```

Program 10.2.10: Calculate the BMI (body mass index) for each weight of each person

```
<TAB!
NAME, LENGTH, (AGE, WEIGHT l)l
Klaus 1.68      18   61
                30   65
                56   80
                61   75
Kathi 1.70      18   55
                40   70
!TAB>
BMI:= WEIGHT : LENGTH : LENGTH
rnd 2
```

Result (tab)

```
NAME ,LENGTH, (AGE, WEIGHT, BMI l) l
Klaus 1.68      18   61       21.61
                30   65       23.03
                56   80       28.34
                61   75       26.57
Kathi 1.70      18   55       19.03
                40   70       24.22
```

Note that the given formula is applied not only to the rows where a length is written, but also to the following rows. This is possible because the table has a certain scheme and our system can understand the scheme.

10.3 Restructuring (gib)

The restructuring operation (stroke) allows to restructure any tabment into another arbitrary tabment only by specifying the scheme or the TT of the target tabment. Additionally, aggregations, elimination of duplicates, union, sorting and certain joins can be realized.

Program 10.3.1: Illustrate the collection symbols

```
aus students.tab
gib FACm,FACb,FACl,FACm-,FACb-,FACl-,FAC?
```

Result

```
FACm ,FACb ,FACl ,FACm- ,FACb- ,FACl- ,FAC?
Art   Art   Math  Sport  Sport  Sport  Math
Infor Art   Infor Math   Math   Art
Math  Infor Infor Infor  Infor  Art
Sport Infor Art   Art    Infor  Infor
      Math  Art          Art    Infor
      Sport Sport        Art    Math
```

The STID segments (type: (STID, NAME, LOCATION?, STIP, FAC)) are inserted one after another into each of the given FAC collections. COURSE and PROJ segments are ignored. For clarity purposes, we have kept the collection symbols of the gib clause.

Program 10.3.2: Sort students by FAC and NAME.

```
aus students.tab
gib FAC,NAMEb m
```

Result (tabh)

```
FAC,  NAMEb m
Art   Käthe
      Ulrike
Infor Clara
      Sophia
Math  Ernst
Sport Claudia
```

STID segments are inserted first into the FAC level and then deeper into the NAME level, segment by segment. COURSE and PROJ segments are not touched anymore.

Program 10.3.3: Sort students by FAC and NAME, resulting in a flat table

```
aus students.tab
gib FAC,NAME m
```

Result (tab)

```
FAC , NAME m
Art   Käthe
Art   Ulrike
Infor Clara
Infor Sophia
Math  Ernst
Sport Claudia
```

If we replace m with b, the result elements do not change.

Program 10.3.4: Sort the faculties downwards by BUDGET and secondly by Student Capacity.

```
aus facs.tab
gib BUDGET,STUDCAPACITY,FAC m-
```

Result (tab)

```
BUDGET,STUDCAPACITY,FAC m-
```

```
10000 500          Infor
 8000 150          Sport
 2000 600          Art
 1000 200          Math
 1000 10           Philo
```

Program 10.3.5: Sort the faculties by budget and additionally by student capacity. (two independent sortings of one table).

```
aus facs.tab
gib BUDGET,FAC m-,(STUDCAPACITY,FAC m-)
```

Result (tab)

```
BUDGET,FAC m-,(STUDCAPACITY,FAC m-)
10000  Infor   600          Art
 8000  Sport   500          Infor
 2000  Art     200          Math
 1000  Philo   150          Sport
  000  Math     10          Philo
```

Program 10.3.6: Pack each student's exam data by department.. (Re-group already grouped data).

```
aus students.tab
gib FAC,(COURSE,MARK b)m
```

Result (tab)

```
FAC ,(COURSE ,  MARK b) m
Art   Apel      1
      Repin     1
Infor Algebra   3
      Databases 1
      Databases 1
      OCaml     2
      Otto      1
Math  Algebra   1
      History   1
      Logic     2
Sport Psycho    2
      Ski       1
```

Here the STID segments are inserted into the FAC level. They cannot be inserted deeper because they contain neither COURSE nor MARK values. The corresponding exams bags are then initially always empty. Then each COURSE segment ((COURSE, MARK) pair) is extended by its parent STID segment. These extended segments can be inserted step by step into the corresponding bgs. The extended segment has the type: (STID, NAME, LOC?, STIP, FAC, COURSE, MARK) PROJ segments are not needed.

Program 10.3.7: (Special selection with gib clause) Give all students, for which an LOC entry exists, with this entry. Give additionally the given collection for comparison purposes.

```
aus students.tab
gib NAME,LOC m,(NAME,LOC? m)
```

Result (tab)

```
NAME ,  LOC m,  (NAME ,  LOC? m)
Clara   Oehna    Clara   Oehna
Claudia Berlin   Claudia Berlin
Ernst   Oehna    Ernst   Oehna
```

```
Käthe   Gerwisch Käthe   Gerwisch
Sophia  Berlin   Sophia  Berlin
                 Ulrike
```

In the first set, the user requests complete pairs. Since no pair exists for Ulrike, she cannot appear in the first result.

Program 10.3.8: (Selection with gib clause only.) Specify all students with non-empty exam collections.

```
aus students.tab
gib STID,NAME,FAC,COURSE,MARK m
gib STID,NAME,FAC,(COURSE,MARK m)m
```

Result (tab)

```
STID, NAME ,   FAC ,(COURSE ,   MARK m) m
1111  Ernst    Math  Algebra    1
                     History    1
                     Logic      2
2222  Sophia   Infor Algebra    3
                     Databases  1
                     Otto       1
3333  Clara    Infor Databases  1
                     OCaml      2
5555  Käthe    Art   Apel       1
                     Repin      1
6666  Claudia  Sport Psycho     2
                     Ski        1
```

Intermediate result after the first gib clause:

```
1111  Ernst   Math  Algebra   1
1111  Ernst   Math  History   1
1111  Ernst   Math  Logic     2
2222  Sophia  Infor Algebra   3
2222  Sophia  Infor Databases 1
2222  Sophia  Infor Otto      1
3333  Clara   Infor Databases 1
3333  Clara   Infor OCaml     2
5555  Käthe   Art   Apel      1
5555  Käthe   Art   Repin     1
6666  Claudia Sport Psycho    2
6666  Claudia Sport Ski       1
```

To get the intermediate result, STID segments are tried to be inserted first. This is not possible because there is no exams-data on this level. Then again, each COURSE segment is extended by its first level parent data. This data is inserted exam by exam and student by student.

Program 10.3.9: For each name, output the "first" MARK entry or "null value" if no check entry is present. Print the other collections for comparison purposes.

```
aus students.tab
gib (NAME,MARK? m),(NAME,MARK m),(NAME,MARK b),(NAME,MARKb m)
```

Result (tabh)

```
NAME,   MARK? m, (NAME, MARK m),   (NAME, MARK b),   (NAME, MARKb m)
Clara   1          Clara   1         Clara   1         Clara   1 2
Claudia 2          Clara   2         Clara   2         Claudia 1 2
```

```
Ernst    1          Claudia 1          Claudia 1          Ernst    1 1 2
Käthe    1          Claudia 2          Claudia 2          Käthe    1 1
Sophia   3          Ernst   1          Ernst   1          Sophia   1 1 3
Ulrike              Ernst   2          Ernst   1          Ulrike
                    Käthe   1          Ernst   2
                    Sophia  1          Käthe   1
                    Sophia  3          Käthe   1
                                       Sophia  1
                                       Sophia  1
                                       Sophia  3
```

STID segments can be inserted in the first and last collection, since only names are required.

Program 10.3.10: (Restructuring) Reverse the given structuring. I.e., swap COURSE and NAME.

```
aus students.tab
gib COURSE,(NAME,MARK b)m
```

Result (tab)

```
COURSE , (NAME ,  MARK b) m
```

```
Algebra   Ernst   1
          Sophia  3
Apel      Käthe   1
Databases Clara   1
          Sophia  1
History   Ernst   1
Logic     Ernst   2
OCaml     Clara   2
Otto      Sophia  1
Psycho    Claudia 2
Repin     Käthe   1
Ski       Claudia 1
```

Here, an attempt is first made to insert the STID segments. Since no COURSE attribute exists, they cannot be inserted. Therefore, the extended COURSE segments are inserted first at the COURSE level and then at the NAME level.

Program 10.3.11: (Restructuring with additional tags) Reverse the given structuring by changing COURSE from inner to outer collection and NAME from outer to inner. Create additional tuple and sub-tuple tags.

```
aus students.tab
gib COURSES
    COURSES     = COURSETUPLEm
    COURSETUPLE = COURSE,EXAMSTUPLEb
    EXAMSTUPLE  = NAME,MARK
```

Result (ment)

```
<COURSES>
  <COURSETUPLE>
    <COURSE>Algebra</COURSE>
    <EXAMSTUPLE>
      <NAME>Ernst</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
    <EXAMSTUPLE>
      <NAME>Sophia</NAME>
      <MARK>3</MARK>
```

```
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Apel</COURSE>
    <EXAMSTUPLE>
      <NAME>Käthe</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Databases</COURSE>
    <EXAMSTUPLE>
      <NAME>Clara</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
    <EXAMSTUPLE>
      <NAME>Sophia</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>History</COURSE>
    <EXAMSTUPLE>
      <NAME>Ernst</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Logic</COURSE>
    <EXAMSTUPLE>
      <NAME>Ernst</NAME>
      <MARK>2</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>OCaml</COURSE>
    <EXAMSTUPLE>
      <NAME>Clara</NAME>
      <MARK>2</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Otto</COURSE>
    <EXAMSTUPLE>
      <NAME>Sophia</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Psycho</COURSE>
    <EXAMSTUPLE>
      <NAME>Claudia</NAME>
      <MARK>2</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
```

```
  <COURSETUPLE>
    <COURSE>Repin</COURSE>
    <EXAMSTUPLE>
      <NAME>Käthe</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Ski</COURSE>
    <EXAMSTUPLE>
      <NAME>Claudia</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
</COURSES>
```

Now we want to illustrate the set-theoretic operations union, intersection, and set-difference. Since the STID column in students.tab is already a union, we illustrate the union with the files exams1 and projects1.

Program 10.3.12: Construct the union of two files, where each value of each file should appear in the result

```
aus examen1.tab,projects1.tab # a pair of tables
gib STIDb
```

Result (tabh, width 50))

```
STIDb
1111 1111 1111 1111 1111 1111 2222 2222 2222 2222
2222 2222 3333 3333 3333 3333 5555 5555 5555 5555
6666 6666 6666 6666
```

If we replace b with m in the gib statement, duplicates are eliminated.
Result

```
STIDl
1111 2222 3333 5555 6666
```

If we want to know from which file each STID comes from, we can add corresponding information

```
aus examen1.tab,projects1.tab # a pair of tables
gib STID,COURSE?,PROJ? b
```

Result

```
STID,COURSE?,  PROJ? b
1111           Fritz
1111           Otto
1111 Algebra
1111 History
1111 Logic
2222           Ghandi
2222           Ming
2222           Otto
2222 Algebra
2222 Databases
2222 Otto
3333 Databases
3333 OCaml
4444            Monet
5555 Monet
5555 Apel
```

```
5555 Repin
6666          Matthes
6666          Witt
6666 Psycho
6666 Ski
```

Program 10.3.13: Construct the intersection of two files with different schemas.

```
aus exams1.tab,projects1.tab
gib STID,COURSE?,PROJ? m
gib STID,COURSE,PROJ m
# This restructuring can also be realized through selections
gib STIDm
```

Result (tabh, width 50))

```
STIDm
1111 2222 5555 6666
```

Intermediate result after the first gib statement

```
STID, COURSE?, PROJ? m
1111 Algebra   Fritz
2222 Algebra   Ghandi
3333 Databases
4444           Monet
5555 Apel      Monet
6666 Psycho    Matthes
```

Program 10.3.14: Set difference: Specify all STIDs of examen1.tab that are not contained in projekte1.tab.

```
aus exams1.tab
rename STID ! STUDID
,projects1.tab # in turn results in a tuple (pair) of tables
sans STUDID in STIDm
gib STUDIDm
```

Result (tab)

```
STUDIDm
3333
```

Program 10.3.15: like 10.3.14, but with nested query

```
aus exams1.tab
sans STID in begin projects1.tab;; gib STIDm end
gib STIDm
```

Result (tab)

```
STUDIDm
3333
```

Program 10.3.16: (Grouping with Aggregation) Calculate the number of students and the number for each faculty. Sort the students by FAC and NAME.

```
aus students.tab
gib CNT,(FAC,CNT,(NAME,STID b)m)
    CNT:= STID! ++1
```

Result (tab)

```
CNT,  FAC , CNT2, (NAME ,  STID b) m
6     Art   2       Käthe    5555
                    Ulrike   4444
```

```
       Infor 2       Clara   3333
                     Sophia  2222
       Math  1       Ernst   1111
       Sport 1       Claudia 6666
```

Program 10.3.17: (Restructuring with Aggregation) Give the total of all scholarships and the total for each course. Sort the records by course.

```
aus students.tab
gib SU,(COURSE,SU m)
    SU:= STIP! ++
```

Result (tab)

```
SU ,(COURSE ,  SU2 m)
2950 Algebra   900
     Apel      600
     Databases 850
     History   500
     Logic     500
     OCaml     450
     Otto      400
     Psycho    600
     Repin     600
     Ski       600
```

It is interesting to note here that the ++ of the inner SU values is generally larger than the outer SU value. This is due to the fact that a particular course usually occurs in more than one student record.

Program 10.3.18: Search the name of the student with ID 2222

```
aus  students.tab
avec STID = 2222
gib  NAME
```

Result (tab)

```
NAME
Sophia
```

Program 10.3.19: Divide the students of the two faculties of computer science and art into two independent tables.

```
aus students.tab
STIDINFOR:= STID if FAC=Infor
STIDART  := STID if FAC=Art
gib STIDINFOR,NAME,LOC? m,(STIDART,NAME,LOC? m)
```

Result (tab)

```
STIDINFOR, NAME,  LOC? m, (STIDART, NAME,  LOC? m)
2222       Sophia Berlin  4444      Ulrike
3333       Clara  Oehna   5555      Käthe  Gerwisch
```

The concept of hierarchical path is important for all operations. Its definition is based on "narrow" schemes. All collection symbols except '?' are real collection symbols.

A **schema s is narrow** if for any 2 real collection symbols c and c' holds, either c is contained in c' or c' is contained in c. Fields f1 and f2 of a schema s are on a **hierarchical path** (**HP for** short) **with respect to s** if the schema formed by forgetting all fields except f1and f2 is narrow.

X,Ym,Zm m is not narrow, but X,Y?,Z? m is. PROJ and COURSE are in NAME, (COURSE,MARK m),(PROJ,HOURS m)m, not on a hierarchical path; unlike PROJm and COURSE. This is visible in the graphical representation of the schema.

```
                m
                |
(NAME, m,                  m)
        |                  |
(COURSE,MARK)        (PROJ,HOURS)
```

Program 10.3.19: Put simply two fields that are not on one HP onto one HP

```
aus students.tab
gib COURSE,PROJ m # in any case empty
```

Result (tabh)

```
COURSE, PROJ l
```

Program 10.3.20: Sort and group the students, who have taken an Algebra COURSE by their corresponding grades and sort them by name and print all their projects.

```
aus  students.tab
avec COURSE=Algebra
gib  NAME,MARK?,(PROJ,HOURS m)m
gib  MARK,(NAME,(PROJ,HOURS m)b) m
```

Result

```
MARK, (NAME, (PROJ,  HOURS m) b) m

1      Ernst  Fritz  4
              Otto   2
3      Sophia Ghandi 5
              Ming   4
              Otto   6
```

Although PROJ and MARK are not on one HP, the project collection is not empty. This could be realized by a somewhat subtle formulation with 2 gib statements. We will get to know a more easily understandable formulation of such requests in the following.

10.4 A simple join and nested queries

The horizontal merging or joining of the information of two tables is called a "join". In our approach, joining two flat tables is not necessarily a flat table. We do not need an additional join operation. Meaningful structures can be created with :=.

Program 10.4.1: Add exam data to student data ..

```
aus students1.tab
EXAMS:= exams1.tab at FAC
```

Initial part of the result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(STID, COURSE ,  MARK m) m

1111  Ernst   Oehna    500   Math  1111  Algebra   1
                                   1111  History   1
                                   1111  Logic     2
                                   2222  Algebra   3
                                   2222  Databases 1
                                   2222  Otto      1
                                   3333  Databases 1
                                   3333  OCaml     2
```

```
                                   5555  Apel      1
                                   5555  Repin     1
                                   6666  Psycho    2
                                   6666  Ski       1
2222  Sophia  Berlin   400   Infor 1111  Algebra   1
                                   1111  History   1
                                   1111  Logic     2
                                   2222  Algebra   3
                                   2222  Databases 1
                                   2222  Otto      1
                                   3333  Databases 1
                                   3333  OCaml     2
                                   5555  Apel      1
                                   5555  Repin     1
                                   6666  Psycho    2
                                   6666  Ski       1
3333  Clara   Oehna    450   Infor 1111  Algebra   1
                                   1111  History   1
                                   1111  Logic     2
                                   2222  Algebra   3
                                   ...
...
```

The result contains 5*10=50 subtuples. To get the 10 desired subtuples, we need to add a condition.

Program 10.4.2: Add all corresponding exam records to each student record ("structured left outer join").

```
aus students1.tab
EXAMS:=exams1.tab at FAC
avec COURSE! STUDENTS1/STID=EXAMS/STID
```

Result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(STID, COURSE ,  MARK m) m
1111  Ernst   Oehna    500   Math  1111  Algebra   1
                                   1111  History   1
                                   1111  Logic     2
2222  Sophia  Berlin   400   Infor 2222  Algebra   3
                                   2222  Databases 1
                                   2222  Otto      1
3333  Clara   Oehna    450   Infor 3333  Databases 1
                                   3333  OCaml     2
4444  Ulrike           400   Art
5555  Käthe   Gerwisch 600   Art   5555  Apel      1
                                   5555  Repin     1
6666  Claudia Berlin   600   Sport 6666  Psycho    2
                                   6666  Ski       1
```

If we want to omit Ulrike, we just have to omit the level identifier 'COURSE:'. Each tabment with the name xyz.tab has the outermost tag XYZ. Therefore, the above extension results in the following TT:

TT (tab type of the result)

```
TABMENT! STUDENTS1
STUDENT1! STID,NAME,LOC?,STIP,FAC,EXAMS m
EXAMEN! STID,COURSE,MARK m
MARK STIP STID! ZAHL
COURSE FAC LOC NAME! TEXT
```

This TT allows accurate specification of column names despite duplicate name occurrences. EXAMS/COURSE is the same as COURSE, because COURSE appears only once on the right side of the TT.

In addition, STUDENTS1 is on the right side of EXAMS, so the tag path STUDENTS1/EXAMS/COURSE is also identical to COURSE. However, the "tag path" STID does not specify exactly, since it occurs twice. STUDENT1/STID is the student identifier of the studenten1 table and EXAMEN/STID=STUDENTEN1/EXAMEN/STID of the exam table.

In a marker path X/Y/Z, Z must occur on the right side of Y and Y must occur on the right side of X in TT. X is the paternal marker of Y and Y is the paternal marker of Z. There is no tag between X and Y and Y and Z (in the XML or ment representation). If we don't know all the intermediate tags, we can also use the X//Z notation. In this case, there can be any number of tags between X and Z. That is, this tag path corresponds to a complete tag path X/X1/X2/.../Xn/Z for matching tags X1,...,Xn. Therefore STUDENTS1//COURSE describes COURSE in the same way as the full tag path STUDENTS1/EXAMS/COURSE.

Program 10.4.3: Program with nested query

```
aus students1.tab
EXAMS:=begin aus exams1.tab;;avec STID=STID~ end at FAC
```

Result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(STID, COURSE ,  MARK m) m
1111  Ernst   Oehna    500   Math  1111  Algebra    1
                                   1111  History    1
                                   1111  Logic      2
2222  Sophia  Berlin   400   Infor 2222  Algebra    3
                                   2222  Databases  1
                                   2222  Otto       1
3333  Clara   Oehna    450   Infor 3333  Databases  1
                                   3333  OCaml      2
4444  Ulrike           400   Art
5555  Käthe   Gerwisch 600   Art   5555  Apel       1
                                   5555  Repin      1
6666  Claudia Berlin   600   Sport 6666  Psycho     2
                                   6666  Ski        1
```

Nested queries are contained in begin and end. If we want to refer to a column name outside the inner query, we must add a "~". Therefore STID~ is the identifier of STID of students1.

Program 10.4.4: Attempts to generate the given student table from three given flat relations..

```
aus students1.tab
PR:=begin aus projects1.tab
          avec STID=STID~
          gib PROJ,HOURS m end at FAC
EX:=begin aus exams1.tab
          avec STID=STID~
          gib COURSE,MARK m end at FAC
```

Result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(COURSE ,  MARK m ),(PROJ ,   HOURS m) m
1111  Ernst   Oehna    500   Math  Algebra   1          Fritz    4
                                   History   1          Otto     2
```

```
                                   Logic     2
2222  Sophia  Berlin   400   Infor Algebra   3           Ghandi   5
                                   Databases 1           Ming     4
                                   Otto      1           Otto     6
3333  Clara   Oehna    450   Infor Databases 1
                                   OCaml     2
4444  Ulrike           400   Art                         Monet   10
5555  Käthe   Gerwisch 600   Art   Apel      1           Monet   20
                                   Repin     1
6666  Claudia Berlin   600   Sport Psycho    2           Matthes  8
                                   Ski       1           Witt    12
```

The result corresponds to students.tab.

Program 10.4.5: Generate a table with three nested levels.

```
aus facs.tab
weg STUDCAPACITY
ST:=begin aus students1.tab
    avec FAC=FAC~
    weg  FAC end at BUDGET
EX:=begin aus exams1.tab
    avec STID=STID~
    weg  STID end at STIP
```

Result (tab)

```
FAC   ,DEAN  , BUDGET,  (STID, NAME  , (LOC?    ,STIP,((COURSE  ,  MARK m) m) m
Art   Sitte    2000     4444  Ulrike            400
                        5555  Käthe   Gerwisch 600     Apel      1
                                                       Repin     1
Infor Reichel 10000     2222  Sophia  Berlin   400     Algebra   3
                                                       Databases 1
                                                       Otto      1
                        3333  Clara   Oehna    450     Databases 1
                                                       OCaml     2
Math  Dassow   1000     1111  Ernst   Oehna    500     Algebra   1
                                                       History   1
                                                       Logic     2
Philo Hegel    1000
Sport Streich  8000     6666  Claudia Berlin   600     Psycho    2
                                                       Ski       1
```

If we want to delete only a few columns, we can use the weg (away) clause instead of a gib clause. We notice that we get a structure with nesting depth 3, although the deepest nesting level in the program is 2.

10.5 A user-friendly join (igib)

Through Example 10.3.20, it has become clear that the problem of loading data onto an HP when it is not yet on an HP in the source structure can be solved in some situations with an additional gib statement without using the Cartesian product. This problem is even more important when we consider a given relational database with flat structures. In a tuple of such files, nothing is on an HP except the fields that are in the same table. Therefore, an ordinary gib statement is not very expressive. Relational systems solve this problem with joins. But the join is related to the Cartesian product. Moreover, join conditions have to be used. In [Gol08] experiments with students were described. They showed that missing join conditions are the most common semantic SQL error. If we use both constructs of this section, the join conditions generally do not need to be written. Moreover, the igib construct is not based on the Cartesian product at all. In the first part of this section, we present some typical queries for igib. It is easy to use igib, but its definition seems to be a bit more complicated than its application.

Program 10.5.1: Give the very good exams and the time-intensive projects for all students who do not live in Gerwisch, who completed a COURSE with a 1, and who have the time-intensive projects. Group the students by place of residence.

```
aus  students1.tab,exams1.tab,projects1.tab
sans LOC=Gerwisch
avec MARK=1
avec HOURS>2
igib LOC,(NAME,(COURSE,MARK m),(PROJ,HOURS m)b)m
```

Result (tab)

```
LOC , (NAME , (COURSE ,  MARK l), (PROJ ,  HOURS m) b) m
Berlin Claudia Ski       1         Matthes  8
                                   Witt    12
       Sophia  Databases 1         Ghandi   5
               Otto      1         Ming     4
                                   Otto     6
Oehna  Ernst   Algebra   1         Fritz    4
               History   1
```

The restructuring by igib contains a preceding natural selection natsel. This can also be used independently of igib. natsel selects until 2 columns with the same name (here only STID) have the same values.

```
aus  students1.tab,exams1.tab,projects1.tab
natsel
```

Corresponding intermediate result after natsel:

```
STID, NAME,    LOC?,     STIP, FAC   m,(STID, COURSE,    MARK  m),(STID, PROJ,    HOURS m)
1111  Ernst    Oehna     500   Math     1111  Algebra    1          1111  Fritz     4
2222  Sophia   Berlin    400   Infor    1111  History    1          1111  Otto      2
5555  Käthe    Gerwisch  600   Art      1111  Logic      2          2222  Ghandi    5
6666  Claudia  Berlin    600   Sport    2222  Algebra    3          2222  Ming      4
                                        2222  Databases  1          2222  Otto      6
                                        2222  Otto       1          5555  Monet    20
                                        5555  Apel       1          6666  Matthes   8
                                        5555  Repin      1          6666  Witt     12
                                        6666  Psycho     2
                                        6666  Ski        1
```

Program 10.5.2: Group and sort student names with bad grades by location and faculty and output the students' bad courses.

```
aus facs.tab,students1.tab,exams1.tab
avec MARK>2
igib LOC,FAC,DEAN,NAME,(COURSE,MARK m) m
```

Result (tab)

```
LOC,    FAC,   DEAN,    NAME, (COURSE, MARK   m) m
Berlin Infor Reichel Sophia Algebra 3
```

Program 10.5.3: Give out all students from Oehna with dean, courses and projects..

```
aus  students.tab,facs.tab
avec LOC=Oehna
igib STID,NAME,FAC,DEAN,COURSEm,PROJm m
```

Result (tab)

```
STID, NAME, FAC,  DEAN,   COURSEm,  PROJm  m
1111  Ernst Math  Dassow  Algebra   Fritz
                          History   Otto
                          Logic
3333  Clara Infor Reichel Databases
                          OCaml
```

Program 10.5.4: Add the instructor column to the courses of the students of the computer science faculty.

```
aus  students.tab,courses.tab
avec FAC=Infor
igib NAME,LOC?,(COURSE,TEACHER,MARK m),PROJm m
```

Result (tab)

```
NAME , LOC?, (COURSE ,  TEACHER, MARK m),PROJm m
Clara  Oehna  Databases Saake   1
Sophia Berlin Algebra   Reichel 3        Ghandi
              Databases Saake   1        Ming
              Otto      Benecke 1        Otto
```

Program 10.5.5: Find all students from large faculties who have a good mark in algebra. FAC ,(LOC ,NAMEb m) m . Structure students by FAC and LOC, and sort them by NAME.

```
aus  facs.tab,students1.tab,exams1.tab
avec STUDCAPACITY>300
avec MARK<4 & COURSE=Algebra
igib FAC,(LOC,NAMEb m)m
```

Result (tab)

```
FAC ,(LOC   ,NAMEl m) m
Infor Berlin Sophia
```

11 Queries to Wikipedia (keys)

Like other Wikipedia, the German Wikipedia represents a great treasure of knowledge about Germany and the world. After the English Wikipedia, it is the largest in the world. It already has 2.6 million entries and the number, scope and quality of the content are constantly being improved. Today, most queries to Wikipedia are of a simple nature. Give me the entry with the key xyz. These queries are answered quickly and are nicely laid out. If the key does not exist, a full text search is performed.

Since Wikipedia's documents are very well structured, many more interesting queries could be realized. In addition, many simple queries can be enabled if many documents are extended by further metadata (e.g. infoboxes) in a suitable way. Metadata is data about data. For example, the "column name" LENGTH (or AREA) is already contained in each RIVER (or COUNTRY) document. Therefore, it would be possible to extract the 10 largest rivers in the world or Europe by a simple query, if appropriate query facilities exist. These are typical database queries. Next to images, structured text takes up the largest amount in Wikipedia. This means that we need a language that can combine databases and document queries in a user-friendly way. It must have finely granulated selection capabilities and the ability to select parts of documents and combine these parts into new documents. Also, Wikipedia contains a significant amount of numerical data. Therefore, it must also have facilities for computation. For example, since the age of a person is usually not included in Wikipedia, calculation capabilities are necessary. In the above case, only the difference between the year of death or the current date and the birth must be realized.

Compared to the other documents of the Internet (HTML), Wikipedia is very well structured. Nevertheless, Wikipedia documents do not have such a fixed type as XML documents. It should be achieved that the existing and future documents are further standardized (define suitable infoboxes and adapt documents to them) to ensure that queries can be formulated as simply as possible. This means that in addition to increasing the quantity, the quality should also be improved in the above sense.

The following TT represents approximately corresponding parts of the structure of the German Wikipedia. The user must know this metadata or at least have it available. Otherwise he will not be able to formulate or understand queries. The metadata is written in German to make corresponding queries easier to understand.

A small but an essential part of the metadata of the German Wikipedia.

```
WIKI! TITEL, (ANR, ATITEL, INHALT m), INFOBOX, INFO s
TITEL,ATITEL,INHALT! TEXT
ANR! ONR
BILD! BTITEL, JPG
URL, WIKILINK! WORT
TITEL ATITEL FETT KURSIV BTITEL! TEXT
Each infobox has its own scheme. We give some examples:
STAAT! EINWOHNER, FLAECHE, OFFIZIELLESPRACHEl,...
STADT! STAAT?, BUNDESLAND?, PROVINZ?, LANDKREIS?,
       GEMEINDE?, HOEHE, FLAECHE, EINWOHNER, ….
FLUSS! GEWAESSERKENNZEICHEN?, LAGEm, FLUSSSYSTEM?,
       LAENGE?,EINZUGSGEBIETl,QUELLE,LINKERNEBENFLUSS,
       RECHTERNEBENFLUSSl, GROSSSTAEDTE, MITTELSTAEDTE,…
LAENGE GROSSTAEDTE MITTELSTAEDTE ...! TEXT
```

From the user's point of view, Wikipedia consists of only one file. However, it contains many different objects of different types. The names of the types are written in the INFOBOX column. For Elbe, Rhine and Vistula, ... river is written there. So a condition INFOBOX=River selects all rivers of

Wikipedia. When transferred to databases, this would be a single large file with its own extensive metadata such as LENGTH, SOURCE, LARGE CITIES,..... . The disadvantage is that all major cities of a river are combined in one field. In the program o++o one must therefore first create a repeating group GROSSSTADTl. It would be better if the Wikipediadatabase would already provide a repeating group here. The infobox FLUSS alone has more than 20 column names. With this it is already clear that due to the many different infobox names a myriad of column names exist. This increases the complexity and is therefore a disadvantage. However, this also allows more diverse queries to be realized, so that queries to Wikipedia open up great new possibilities not only for end users, but also for specialists.

Another feature of the above metadata is that the structured texts are not stored recursively. This facilitates text queries.

Since our implementation so far only works on the German Wikipedia, all examples are in German. The following description is based on queries to a part of Wikipedia with about 28'000 struples (structured tuples = complex data sets). Loading the struples took about 60 minutes. In general, we assume that we load the entire Wikipedia into memory. Loading the part of Wikipedia mentioned above required a total memory consumption of 6.9 GByte. We call our part of the German Wikipedia with the word wiki or 'from wiki'.

s is the abbreviation for set. Unlike m, s does not require a unique scheme.

Program 11.1: How many struples (structured tuples=entries) does our Miniwikipedia contain?

```
wiki ++1 '3
```

Result

```
28'819
```

Program 11.2: Wanted is the first part of the table of contents of the city of Magdeburg.

```
wiki
keys [Magdeburg]
gib  TITEL,(ANR,ATITEL m)
avec ANR <=12
```

Result (tab):

```
TITEL    ,(ANR    ,ATITEL l)
Magdeburg 0       Einleitung
          1       Geographie
          1.1     Schutzgebiete
          1.2     Klima
          1.3     Nachbargemeinden
          1.4     Stadtgliederung
          2       Geschichte
          2.1     Bedeutung und Herkunft des Namens
          2.2     Ur- und Frühgeschichte
          2.3     Mittelalter
          2.4     Frühe Neuzeit
          2.5     19. Jahrhundert
          2.6     Weimarer Republik und Nationalsozialismus
          2.7     Nachkriegs- und DDR-Zeit
          2.8     1990 bis zur Gegenwart
          2.9     1200. Stadtjubiläum
          2.10    Eingemeindungen
```

```
          2.11     Bevölkerungsentwicklung
          3        Religionen
          3.1      Kirchengeschichte
          3.2      Evangelische Kirchen
          3.3      Römisch-katholische Kirche
          3.4      Zeugen Jehovas
          4        Politik
          4.1      Stadtrat
          4.2      Oberbürgermeister seit 1808
          4.3      Wappen, Flagge und Dienstsiegel
          4.4      Städtepartnerschaften
          4.5      Stadtkampagne
          5        Kultur und Sehenswürdigkeiten
          5.1      Bauwerke
          5.1.1    Sakralbauten
          5.1.1.1  Altstadt und alte Neustadt
          5.1.1.2  Außenbezirke
          5.1.2    Festungsanlagen
          5.1.3    Profanbauten und weitere Bauwerke
          5.1.4    Denkmäler und Skulpturen
          5.2      Friedhofsanlagen
          5.3      Plätze und Straßen
          5.4      Brunnen
          5.5      Brücken
          5.6      Museen
          5.7      Galerien
          5.8      Theater und Oper
          5.9      Kabarett
          5.10     Freizeit und Tourismus
          5.11     Zoologischer Garten
          5.12     Parks und Gärten
          5.13     Veranstaltungsorte
          5.13.1   Nachtleben
          5.14     Sport
          5.15     Regelmäßige Veranstaltungen
          5.15.1   Frühling
          5.15.2   Sommer
          5.15.3   Herbst
          5.15.4   Winter
          5.16     Musik
          5.17     Kulinarische Spezialitäten
          5.18     Große Einkaufsmöglichkeiten
          5.19     Stolpersteine
          5.20     Halbkugeln
          6        Wirtschaft und Infrastruktur
          6.1      Industrie
          6.2      Verkehr
          6.2.1    Schienenverkehr
          6.2.2    Straßenverkehr
          6.2.2.1  Straßennamen
          6.2.3    Öffentlicher Personennahverkehr
          6.2.4    Fahrradverkehr
          6.2.5    Schifffahrt
...
```

Program 11.3 Give information about the nightlife of Magdeburg

```
wiki
keys [Magdeburg]
avec ATITEL=Nachtleben
gib  INHALT     # CONTENT
```

Result (web)

Magdeburg's nightlife consists mainly of dance events in larger discotheques and smaller clubs, in addition to live concerts. A distinctive feature of Magdeburg is that many of these venues are located in former fortresses and industrial facilities that have been vacant since reunification. Some larger discotheques include the Festung Mark, which hosts electronic music events as well as cultural events, and the "Alte Theater" on Jerichower Platz. Also offering an industrial feel is the former factory hall "Factory" in the south of the city, where German and international pop, rock, metal, indie bands regularly play and disco events are held. The nobler clubs of the city include the "Prinzzclub" and the "First", which offer a mix of lounge and club. At 45 years old, the "Studentenclub Baracke" is the oldest club in the city and is located directly on the grounds of Otto von Guericke University. As an equivalent, the "Kiste" exists on the campus of the university hospital for students of the medical faculty. In addition, there are other medium-sized and smaller discos and clubs, such as the "Boys'n'Beats", the "Alte Feuerwache", the "Kunstkantine" or the "Triebwerk". Also worth mentioning are the clubs "Strandbar", modeled after a beach, directly on the Elbe, with one of the first citybeach concepts in Germany, and the "Montego Beachclub" in the Stadtpark Rotehorn with volleyball courts and a large pool. In 2016 and 2017, some discos were closed. For example, in the south of the city until 2016 was the large-capacity discotheque "Music Hall", the former "Funpark", which served special music genres in addition to mainstream genres. In addition, in 2016 the "Discoturm Nautica (Pearl Club)" was closed after the fun club changed operators following insolvency and was rebuilt. Finally, at the beginning of 2017, the "Kulturwerk Fichte", a listed industrial hall from the founding times, where scene parties and other large events were held, was closed[[Ref]][[Ref]]. The Hasselbachplatz at the southern city center has developed into Magdeburg's pub center in recent years. Due to the high frequency during the day, but especially in the evening hours by visitors to the numerous clubs, bars and pubs, the square is classified as a crime hotspot and is monitored by video technology.

Program 11.4 like request 11.3 only a little more efficient

```
wiki
keys [Magdeburg, ["5.13.1"]]
gib  INHALT
```

Result

```
same result as 11.3
```

Program 11.5 Output section 5.15 of Magdeburg with all subsections of any depth.

```
wiki
keys [Magdeburg]
avec ANR like "5.15*"
gib ANR,ATITLE,CONTENT m
```

Result (web)

ANR	ATITLE	CONTENTS
5.15	Regular events	

5.15.1 Spring	The '' Magdeburg Spring Fair'', a three-week shindig at the beginning of spring, is held annually at the fairground "Max Wille" at the Kleiner Stadtmarsch directly on the banks of the Elbe. Since 2010, the RoboCup German Open has been held in Magdeburg's exhibition halls [[Ref]] in March/April. Thousands of visitors follow international teams competing with their robots in various disciplines, including robot soccer. Annually, the ''Day of Thunder'' has been held at Magdeburg's airfield since 2000. In 15 different racing classes, different builds of mopeds, motorcycles, cars and quads compete in a 1/8-mile race. In addition, competitions such as "Best of Show", "Most Beautiful Overall Concept", "Best Paintjob", "Best Interior", "Best of Sound", "Best of Exhaust" and a "dB-Sound-Contest" will be held on a show stage. The Magdeburg Historical Spectacle Spectaculum Magdeburgense in May in the area of the old fortifications is a medieval event. Visitors are entertained by numerous events and activities, including, for example, fakir shows, theater performances, a medieval market and musical sounds from the period. Every year on Ascension Day, the ''Festival of Encounters'' against xenophobia takes place in the Rotehorn Park.
5.15.2 Summer	In addition to the cloister serenades in the cathedral, the nationally known summer open air of the Theater Magdeburg takes place on Magdeburg's cathedral square in July/August. The ''BallonMagie'' days in the Elbauenpark are held annually in August. Several hot air balloons take off at the same time and enrich the Magdeburg sky. Special shapes such as ice cream cones, sausage cans or airships are represented among the balloons. Christopher Street Day, also in August, is a day of celebration for lesbians, gays, bisexuals and transgenders. Demonstrations are held for the rights and against the exclusion of these groups. It takes place in numerous cities in Germany. The parade stretches from Neustadt train station through the city center, Hasselbachplatz to Liebigstraße. Around the historic quarter at Magdeburg Cathedral, the three-day ''Kaiser Otto Fest'' has been held annually since 2011, when buildings and squares such as the Cleve Bastion, the Unser Lieben Frauen Monastery, the Möllenvogteigarten, the Fürstenwall or the cathedral itself become venues for medieval attractions, performances and pageants such as Otto I's coronation as emperor, jousting tournaments, falconry shows and medieval songs. The festival is intended to commemorate the importance of the city of Magdeburg as the cradle of the German nation and European history. The bicycle action day takes place once a year. After a rally to the assembly point, a large bicycle demonstration leads across the city and over the Magdeburger Ring. In this way, the cyclists want to show their colors and advocate for a more bicycle-friendly city[[Ref]]. In 2014, it took place for the fourth time on June 28 s day.
5.15.3 Fall	In September, Magdeburg celebrates the Landeserntedankfest, with over 35,000 [[Ref]];visitors the largest public event of the agricultural profession in Saxony-Anhalt, in the Elbauenpark. In addition, there are the jazz festival DIAGONALE, the literature weeks, an event for literature lovers with many offers and exhibitions, lectures and performances, the art festival Magdeburg, the OMMMA (''Ostmobil-Meeting Magdeburg'') and the ''Magdeburger Herbstmesse'' (formerly ''Herrenmesse''), a three-week carnival held at the beginning of autumn on the "Kleiner Stadtmarsch". In 2010 it celebrated its 1000th anniversary, because it finds its origin in the sacred feast of the Theban Legion of Archbishop Tagino, which was celebrated on 22 September 1010. From the year 1220, the feast of Mauritius and his fellow saints merged with the great Magdeburg fair, which at the time was still held on the cathedral square. Thus, the Magdeburg Autumn Fair is today the oldest folk festival in Germany[[Ref]].

5.15.4 Winter	The biggest event of the year is Magdeburg Christmas Market, with around 135 stalls. It attracts over 1.5 million visitors every year, is held on the Alter Markt and offers many attractions, for example daily live music, a Santa Claus talk, fairy tale performances and the historic Christmas market. It is considered one of the most child-friendly Christmas markets in Germany and is the longest daily open Christmas market in Germany[[Ref]]. In January, the so-called ''Mile of Democracy'' is held annually with over 10,000 visitors, with the Breite Weg to Hasselbachplatz being the venue for this event with numerous actions, information booths, discussion hours and an extensive stage program. It was created to take away the space from the march of right-wing extremists taking place at the same time. These use the anniversary of the air raids on Magdeburg on January 16, 1945 as an occasion for a funeral procession, and to equate the victims with the Holocaust and the murdered in the concentration and extermination camps and thus to trivialize the Nazi mass murders.

Program 11.6 Give the summary information of three major rivers..

```
wiki
keys [Mekong Yangtze Amazon]
avec ANR! ANR=0
gib  TITLE,LENGTH,CONTENT m-
```

Result (web)

TITLE	LENGTH	CONTENTS
Mekong	4350	The '''Mekong''' ([] or []) is a stream in Southeast Asia that crosses six countries. Its length is given as 4350 km to 4909 km. This makes it one of the twelve longest rivers on earth.
Yangtze	6380	The '''Yangtze River''', in short '''Yangtze'''' (, for short:), is the longest river in China. With 6380 kilometers, of which 2800 kilometers are navigable, it is also the longest river in Asia and the third longest river in the world after the Nile and the Amazon. Its headwaters are in the highlands of Tibet in Qinghai. At its mouth into the East China Sea, it carries an annual average of 31,900 m³ of water per second. The Yangtze River plays a major role in the self-image of the Chinese. It divides the country into North and South China and has been the site of numerous important events in Chinese history. These include its crossing by the People's Liberation Army during the Chinese Civil War on April 21, 1949, and the right of Western powers to navigate the river with gunboats, which existed until the middle of the 20th century.
Amazon	6992	The '''Amazon River''' (also , , in Brazil above the confluence of the Rio Negro near Manaus [[Ref]], formerly) is a stream in northern South America. About 300 km south of the equator, it crosses the Amazon basin, framed in the west by the Andes and characterized by tropical rainforest, eastward to the Atlantic Ocean. With an average water flow of 206,000 m³/s, the Amazon is by far the most water-rich river on earth and carries more water at its mouth than the six next smallest rivers combined and about 70 times more than the Rhine[[Ref]]. The river takes its name only from the confluence of its two headwaters, the Marañón and the Ucayali, in Peru, interrupted, however, by the Brazilian section above the city of Manaus called ''Rio Solimões''. The river, which is usually several kilometers wide in Brazil, has a relatively balanced water flow, since the flood phases of the tributaries meet the main stream near

	the equator with a seasonal shift. Nevertheless, it can flood the adjacent forested alluvial areas ("várzea") over a width of up to 60 km. In two main branches it flows through the island world of the almost 200 km wide estuary, which is also connected to the Pará estuary by tidal waters, thus separating the large island of Marajó.

The next query identifies all medium and large cities located on the Spree River. The query result comes relatively fast, because not all 30'000 records have to be searched. This is because keys are used here and not the general avec operation. The titles of the records give direct access to the individual struples. When the query optimization is realized, keys should disappear from the user level again and be replaced by avec.

The two assignments for STADTl are necessary because Wikipedia stores the large and medium-sized cities as comma-separated lists. This means that a GROSSSTAEDTE entry results in a list of STADT. Wikipedia is based on the MariaDB relational database management system. These systems have general problems with lists (repeating groups):

Program 11.7: All large and medium-sized cities on the Spree are searched for in one column.

```
wiki
keys [Spree]
STADTl:=GROSSSTAEDTE  cil at GROSSSTAEDTE
STADTl:=MITTELSTAEDTE cil at MITTELSTAEDTE
gib TITEL,STADTl
```

Result (tabh)

```
TITEL,STADT
Spree Cottbus Berlin Bautzen Spremberg Fürstenwalde/Spree
```

Program 11.8: Determine the heights above sea level of three cities

```
wiki
keys Berlin Stuttgart Heidelberg
gib TITEL,HOEHE l
```

Result (tab)

```
TITEL,     HOEHE l
Heidelberg 114
Stuttgart  247
```

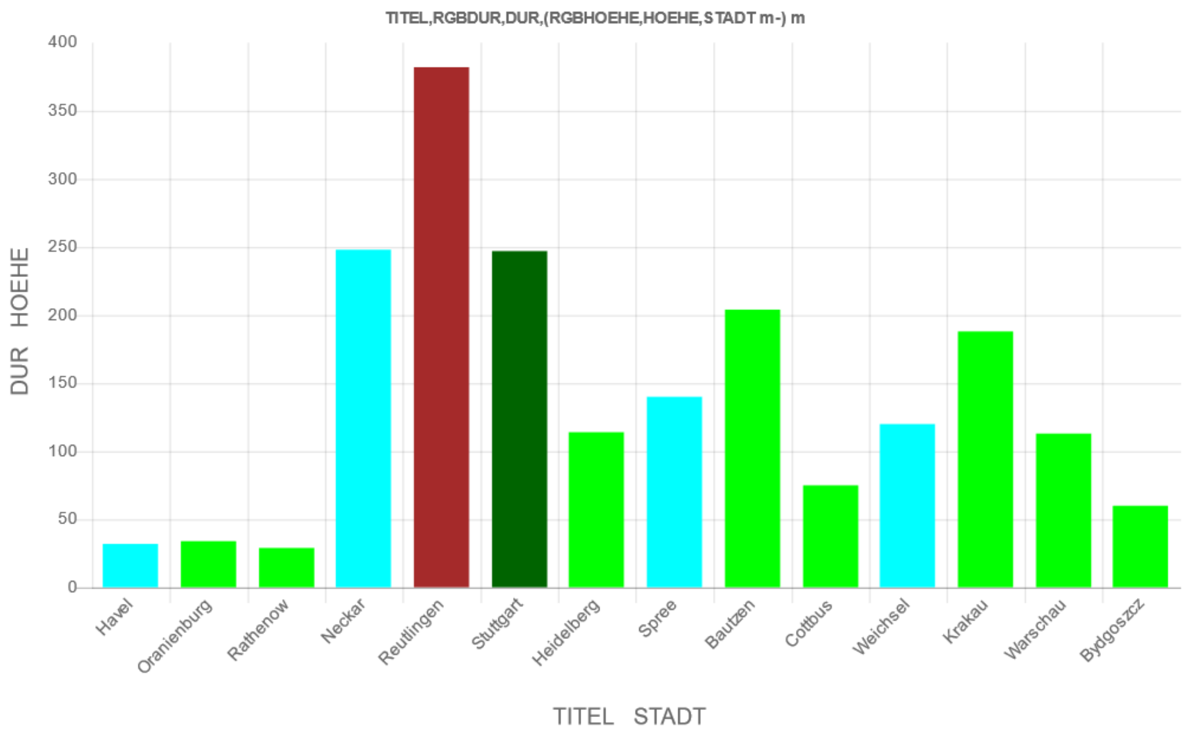
There is an important lesson to be learned from the above query. Even if you know a query language, you do not always get the desired information. Since the above Miniwiki does not contain an entry for Berlin, Berlin cannot appear in the result of this query language.

Program 11.9: Determine for 4 rivers the adjacent cities with their heights above sea level. For each river, determine the average elevation of the cities above sea level.

```
wiki
keys [Neckar Havel Spree Weichsel]
STADTl:=GROSSSTAEDTE  cil at GROSSSTAEDTE
STADTl:=MITTELSTAEDTE cil at MITTELSTAEDTE
gib TITEL,STADTl m
=: $STAEDTE
aus $STAEDTE; gib STADTm
=: $STADTM
aus wiki
```

```
keys $STADTM
gib TITEL,HOEHE? l
rename TITEL!STADT
, $STAEDTE
igib TITEL,(STADT,HOEHE m)m
HOEHE::=HOEHE nthzahl 1
gib TITEL,DUR,(HOEHE,STADT m-) m DUR:=HOEHE!++:
rnd 0
```

Result (diagram columns)



Result (tab)

```
TITEL,    DUR, (HOEHE, STADT  l) l
Havel      32.   34    Oranienburg
                 29    Rathenow
Neckar    248.  382    Reutlingen
                247    Stuttgart
                114    Heidelberg
Spree     140.  204    Bautzen
                 75    Cottbus
Weichsel  120.  188    Krakau
                113    Warschau
                 60    Bydgoszcz
```

The colors in the bar chart are user-defined. Only two additional assignments were required:

```
RGBDUR  :=cyan leftat DUR
RGBHOEHE:=brown if HOEHE>300 !
          darkgreen if HOEHE>220 !
          green leftat HOEHE
```

12 Special restructuring operations (onrs verti hori)

The onrs operation was introduced to provide o++o numbers for solving BOM problems. The given tabment must be of type

```
X1,...,Xn,(Y1, ... Yk m) m
```

where X1 and Y1 are the keys of the respective collections. We assume that both collections are sets. This gives us direct access to the corresponding tuples or subtuples in working memory. We consider an example:

Program 12.1: Build a bill of materials

```
<TAB!
PART,   PROPERTY,    (SUBPART,    COUNT m) m
Bush    cylindrical
Rim     smooth
Polo    modern         Wheel      4
                       Engine     1
                       Body       1
Golf    fast           Wheel      4
                       Body       1
                       Climate    1
                       Engine     1
Piston  light          PistonRing 2
                       Bushing    1
Engine  heavy          Piston     6
                       Screw      8
Wheel   round          Screw      5
                       Tire       1
                       Rim        1
!TAB>
onrs OTTONR ! Golf
```

Result (tab)

```
PART, PROPERTY, (OTTONR, SUBPART,    COUNT l) l
Golf  fast        1        Wheel      4
                  1.1      Tire       1
                  1.2      Screw      5
                  1.3      Rim        1
                  2        Engine     1
                  2.1      Screw      8
                  2.2      Piston     6
                  2.2.1    PistonRing 2
                  2.2.2    Bushing    1
                  3        Climate    1
                  4        Body       1
```

It can be seen that all direct subparts from the Golf are assigned an otto number, which consists of only one number. The engine is one such part. The direct lower parts of the engine (screw and piston) are assigned otto numbers with two numbers. Similarly, the direct lower parts of the piston are assigned otto numbers with 3 numbers. Thus, a non-recursive set without redundancy is formed for the Golf. The table recursion could now be applied to this set to calculate the multiplicity of containing a subpart.

Beside the Inputtabment onrs needs the name of the ONR column (here OTTONR) and one or more parts (here only Golf) for which the ONR resolution is to be made. Accordingly, the program line

onrs STRUK_NR ! [Polo Golf]

is correct and reasonable.

The verti operation can be used to convert certain tuples into lists of pairs. This has the advantage that list operations, such as selecting, can be applied to these lists. This converts the metadata into primary data and inserts a new schema into the existing one. The following file was obtained from an EXCEL table.

`climate_radiation1.tab`

has the scheme:

```
ID,COUNTRY,WIDTH,LENGTH,SEAHEIGHT?,JAN,FEB,MRZ,APR,MAY,JUN,JUL,AUG,
                                            SEP,OCT,NOV,DEC            l
```

It contains 17 columns. Using verti, you can reduce the number of columns to 7 in the following way:

```
aus climate_radiation1.tab
verti MON,RADIATION l:= JAN ..DEC
```

This flat table is transformed into a structured one, in which the radiations are arranged vertically and the months are output in an additional column:

```
ID,           COUNTRY, WIDTH, LENGTH, SEAHEIGHT?,   (MON,RADIATION l) l
BG0001a-Varna Bulgaria 43.21  27.91   44             Jan 63.
                                                     Feb 68.
                                                     Mar 81.
                                                     Apr 87.
                                                     May 88.
                                                     Jun 81.
                                                     Jul 86.
                                                     Aug 100.
                                                     Sep 95.
                                                     Oct 88.
                                                     Nov 66.
                                                     Dec 59.
BG0002a-Shumen Bulgaria 43.283  26.933  242.         Jan 59.
                                                     Feb 68.
                                                     Mar 83.
                                                     Apr 88.
                                                     May 88.
                                                     Jun 81.
                                                     Jul 88.
                                                     . . .
. . .
```

Now you can easily create statistics about RADIATION or easily select specific months, ...

Program 12.2: Arrange the subjects vertically so that they can be merged with the other subjects (for the determination of the report marks).

```
<TABH!
NAME,    LOC,     BORN,    CLASS?, (HOBBY,        HOURS l),MATHl, GERMANl l
Clara    Oehna    12.6.11 4          Riding       5          1 2     3 1 1 1
                                     Chess        1
Claudia Dallgow 14.9.17              Chinese      5
                                     Food         4
Sophia   Dallgow 7.9.13   2          Painting     5          1 2     1 2 1
                                     Wheelturning 4          1 1
                                     Chinese      6
!TABH>
verti SUBJECT,MARKl l:=MATHl .. GERMANl
```

Result (tabh)

NAME	,LOC	,BORN	,CLASS?	,(HOBBY	,HOURS l),	(SUBJECT	,MARKl l) l
Clara	Oehna	12.6.11	4	Riding	5	Math	1 2
				Chess	1	German	3 1 1 1

```
Claudia Dallgow 14.9.17          Chinese      5
                                 Food         4
Sophia  Dallgow 7.9.13  2        Painting     5          Math      1 2 1 1
                                 Wheelturning 4          German    1 2 1
                                 Chinese      6
```

The reverse operation to verti is hori. Here the elementary tag whose values are to become column names must be specified as the second input value.

Program 12.3: Arrange the subjects horizontally.

```
<TABH!
NAME,   LOC,    BORN,    CLASS?,(HOBBY,       HOURS l),(SUBJECT,MARKl l)l
Clara   Oehna   12.6.11  4        Riding       5          Math     1 2
                                  Chess        1          German   3 1 1
Claudia Dallgow 14.9.17           Chinese      5
                                  Food         4
Sophia  Dallgow 7.9.13   2        Painting     5          Math     1 2 1 1
                                  Wheelturning 4          German   1 2 1
                                  Chinese      6
!TABH>
hori SUBJECT
```

Result (tabh)

```
NAME,   LOC?,   BORN?,  CLASS?, (HOBBY,       HOURS  l),MATHl, GERMANl  l
Clara   Oehna   12.6.11 4        Riding       5          1 2     3 1 1
                                 Chess        1
Claudia Dallgow 14.9.17          Chinese      5
                                 Food         4
Sophia  Dallgow 7.9.13  2        Painting     5          1 2 1 1 1 2 1
                                 Wheelturning 4
                                 Chinese      6
```

Just as in the above example the output table is created again by suitable successive application of verti and hori, this can also be achieved in the previous example:

Program 12.4 Application of verti and hori

```
climate_radiation1.tab
verti MONTH,RADIATION l :=JAN .. DEC
hori MONTH
```

Result (meta)

```
TABMENT! CLIMATE_RADIATION1
CLIMATE_RADIATION1! (ID,COUNTRY,WIDTH,LENGTH,SEAHEIGHT?,JAN,FEB,MRZ,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC l)
ID COUNTRY! TEXT
APR AUG WIDE DEC FEB JAN JUL JUN LENGTH MAY SEA HEIGHT MARCH NOV OCT SEP! PZAHL
```

13 Some operations for text processing with o++o (+ -+ cil zil satzl)

The + symbol can also be used to concatenate and manipulate text. Here, too, a small difference is made between TEXT and WORT. Example:

Program 13.1: There are small differences between WORD and TEXT concatenation

```
WORDRESULT:=otto + " o++o"
TEXTRESULT:=otto text + " o++o"
```

Result (tab)

```
WORDRESULT, TEXTRESULT
otto_o++o   otto o++o
```

Since the first input value of WORDRESULT is a word, the result is also of type WORT. The same applies to the second case, where the result is a text. Words cannot contain spaces.

-+ is an operation with 3 input values

The TT of the first input value is retained. Each occurrence of the second input value is replaced by the third.

Program 13.2: -+text

```
<TAB!
X, Y l
1 Today is Monday.
2 Yesterday is Sunday.
!TAB>
-+text "is S" ! "was S"
```

Result (tab)

```
X,Y l
1 Today is Monday.
2 Yesterday was Sunday.
```

Program 13.3:

```
  TEXTPLUS:="Today is a beautiful " + day
  TEXTMINUS:=Thunmmder_weather - "m"
  TEXTMINUSPLUS:="Today is a beaoetiful day." -+text oe ! u
```

Result (tab)

```
TEXTPLUS,                TEXTMINUS,      TEXTMINUSPLUS
Today is a beautiful day Thunder_weather Today is a beautiful day.
```

A function cil which extracts all words from a tabment has also been implemented. An analogous function for sentences (satzl) has been implemented, which so far uses a relatively rudimentary end-of-sentence detection.

Program 13.4: "Coding" a text.

```
"Today is Tuesday. Tomorrow is Wednesday."
zil
sans WORT inmath ["a" "e" "i" "o" "u"]
WORT::="t" if WORT="m"!
       "m" if WORT="t"!
       WORT;; ++text
```

Result (tab)

```
TEXT
Tdy s Tsdy. Ttrrw s Wdnsdy.
```

The cil operation is particularly useful for making queries on documents whose schema is not fully known to the user. With the term "avec Schulze back" he can select all elements of a collection that contain the word "Schulze" and the word "back" without having to know the attribute names in detail. If no selection level is specified, the selection is always made in the topmost collections.

Program 13.5: Illustration of "in" in texts

```
<TAB!
NR,SECTION l
1  Today S. Schulze is in Hong Kong. Tomorrow S. Schulze is in Beijing.
2  S. Schulze is back.
3  S. Meier lives in Magdeburg.
!TAB>
avec (Schulze back) in SECTION # in based on cil
```

Result (tab)

```
NR, SECTION l
2   S. Schulze is back.
```

Program 13.6: Eliminate the dot from numbers in a list.

```
Xl:=1.1 ..15 eb 3 # eb: Exponent base = ^ with swapped exponents
Y:= X zil
sans WORT="."
Y::=Y ++text
```

Result (tab)

```
X,                 Y l
      3.3483695221 33483695221
     10.0451085663 100451085663
     30.1353256989 301353256989
     90.4059770967 904059770967
    271.21793129   27121793129
    813.653793871  813653793871
   2440.96138161   244096138161
   7322.88414484   732288414484
  21968.6524345    219686524345
  65905.9573035    659059573035
 197717.871911     197717871911
 593153.615732     593153615732
1779460.8472       17794608472
5338382.54159      533838254159
```

14 Format with o++o ('3 '4 norm3e norm3m mant rnd)

Analogous to SQL, o++o had initially limited itself to content problems. However, o++o uses much richer structures than SQL. Formatting was then taken over from SVG. Thus one could frame a table, write letters bold or colored, ... Now we have implemented more possibilities. Numbers with a larger mantissa are hard to read if they are not grouped. Since many different variants are used for number representations in the world, we have chosen representations that do not collide with the existing ones as much as possible and are still more readable.

Grouping of digit sequences ('3 '4)

Following the Swiss model, o++o uses the apostrophe to make numbers more readable. Blocks of three are the most important along with blocks of four.

Program 14.1: Improve readability of several numbers by grouping them.
12345678, 1234567.87654 '3 ;1234567890 '4
`Result (tab)`
`ZAHL, PNUMBER, ZAHL`
`12'345'678 1'234'567.876'54 12'3456'7890`

Such representations are created by the (unary) operations '3 and '4. The user can also set the apostrophe arbitrarily, for example to make telephone numbers more readable:

`0176'84'208'408`

Operations that generate such are too complicated, so o++o cannot generate them.

Internationally, both the comma and the point (dot) are used as decimal separators and the point and the comma are also used for grouping. We hope to eliminate this inconsistency through these arrangements.

Exponent first notation (norm10m) and norm10e

PZAHL numbers with long mantissa are not to be grasped fast enough, since the more substantial exponent is indicated only at the end of the string. Furthermore, people think in thousands, millions, billions, ... An exponent 7 or 8 must be recognized as first 10- or 100-million. This way of thinking reflects o++o by allowing only multiples of 3 as exponent. Furthermore, the exponents can also be given first:

6m12.345 (12 million ...)

9m123.4 (123 billion ...)

the old mantissa first notation knows o++o nevertheless. However here also multiples of 3 are used as exponent:

12.3456789e6 (12 million ...)

123.456789e9 (123 billion ...)

These formatting's can be generated by the one-digit (unary) operations norm3m (for the representation with m) and norm3e (for the latter). The 3 expresses that the exponent is a multiple of 3.

Program 14.2: Improve the readability of numbers by normalizing the exponents.
12345678.9 norm3m;12345678.9 norm3e
`Result (tab)`
PZAHL,   PZAHL
6m12.3456789 12.3456789e6

The reduction of the digits (mant)

Most people don't care about the many decimal places when a calculator outputs the square root of 2 or 3 with more than 10 digits. The overload of irrelevant information makes it harder for us to see what is important. Therefore, omitting unnecessary digits (information) is important.

The binary function mant realizes this and converts the result immediately into the m-representation. I.e. the operation norm3m is applied at the same time. The second argument of mant specifies the number of digits desired.

Program 14.3: Reduce the number of digits to four.
12345678.98765 , 1234567890 mant 4
`Result (web)`
6m12.34   9m1.234

You can already see from this example that the mant function is especially important for the cell phone. This makes more columns of a result table visible on the display at the same time.

15 Structured diagrams

With o++o you can easily create diagrams. Once you have created an o++o program, you can use the diagram button to open a new browser window that offers a choice of different diagram types. Column charts are certainly the most commonly used. The following rules apply to diagrams:

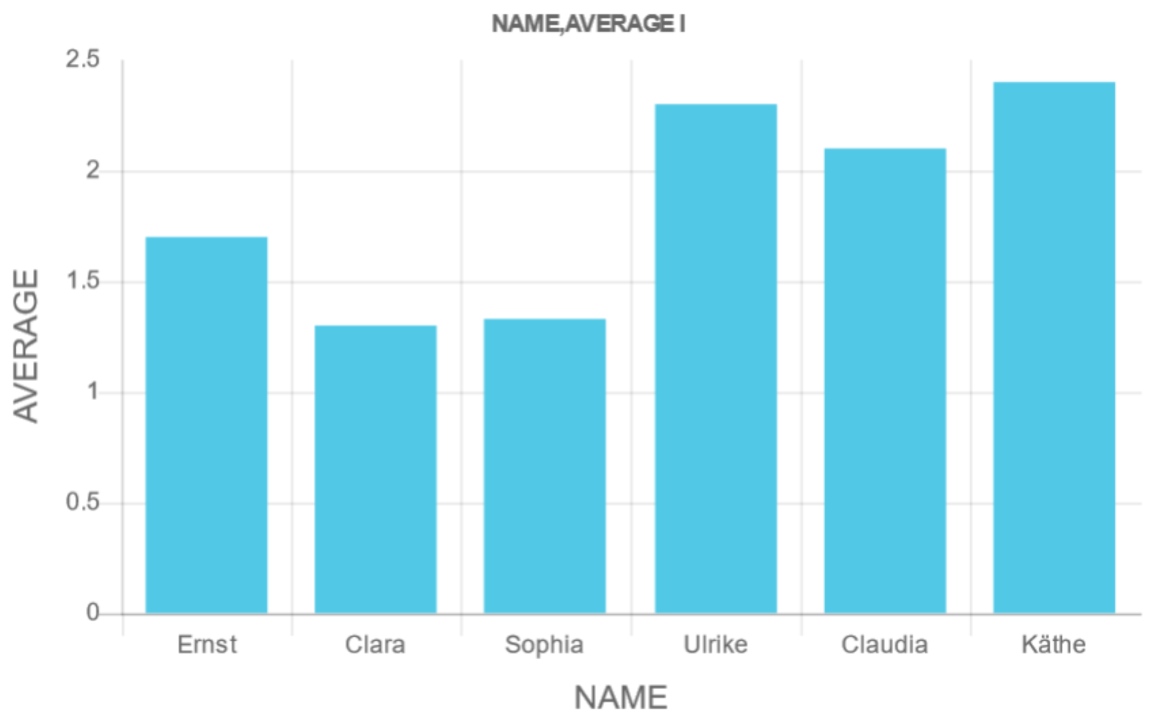
1. TEXTs are converted to words by the system by replacing each space with an underscore.
2. Numeric columns (ZAHL, PZAHL, RATIO) are displayed as columns.
3. The first word column of each hierarchy level is used as the signature for the columns. The other word columns of the level are ignored. If no word column exists, a dash acts as a signature.
4. If no RGB values are given, the system sets default colors. If the user wants to choose the colors, each numeric column must have an RGB column in the same level or higher. If an RGB value is placed directly in front of a number column, it determines the color of the column.
5. If the table to be displayed starts with the column name TITEL, the content of the column is interpreted as the heading of the entire chart.

We already know that a simple list of numbers is an o++o program that can be represented as a diagram. If there is one more word in each line, it serves as a signature:

Program 15.1: Create a column chart with signatures

```
<TAB!
NAME,   AVERAGE 1
Ernst   1.7
Clara   1.3
Sophia  1.33
Ulrike  2.3
Claudia 2.1
Käthe   2.4
!TAB>
```

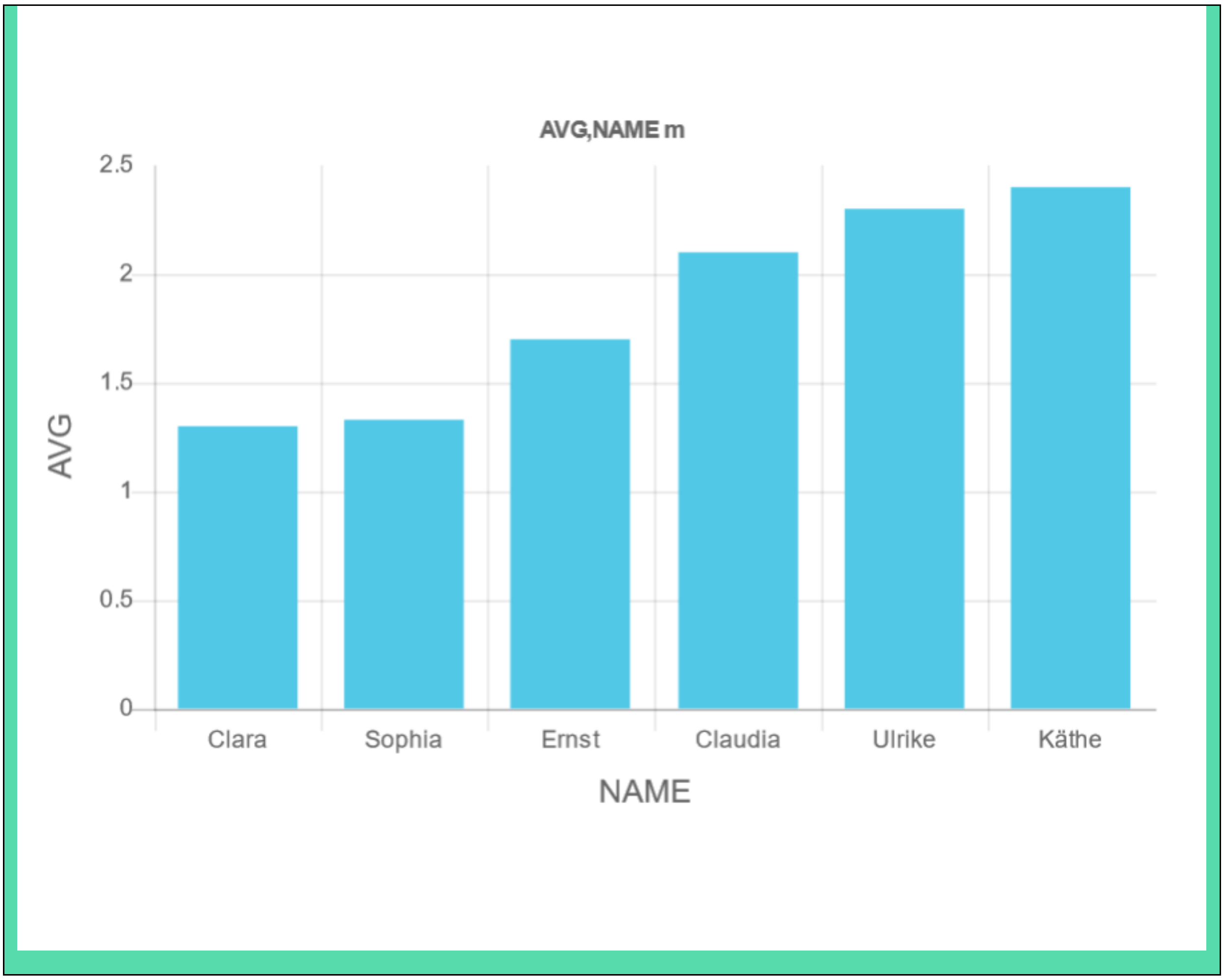
```
Result (Diagram- Säulen (Columns))
```



Program 15.2: Sort the columns with signatures by size

```
<TAB!
NAME,    AVG l
Ernst    1.7
Clara    1.3
Sophia   1.33
Ulrike   2.3
Claudia  2.1
Käthe    2.4
!TAB>
gib AVG,NAME m
```

```
Result (diagram - columns)
```



towers.tab

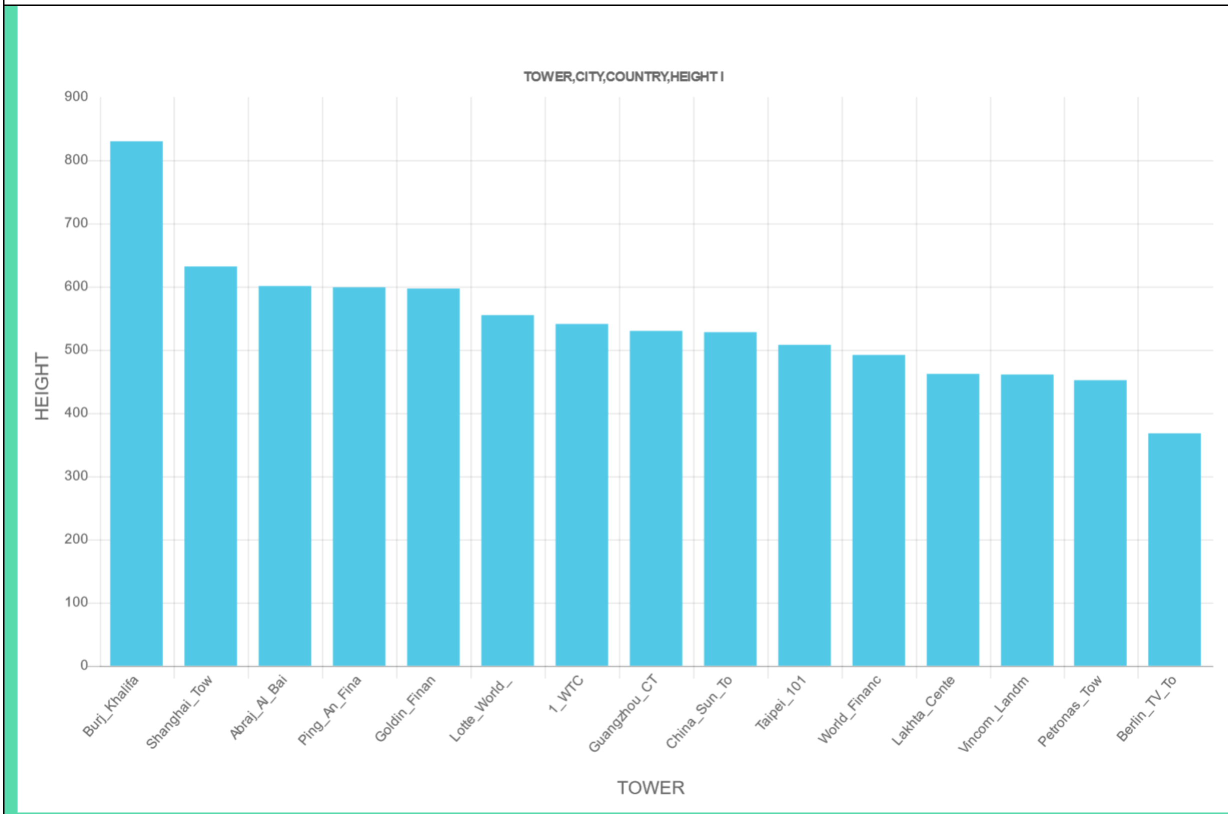
TOWER,	CITY,	COUNTRY,	HEIGHT l
Burj Khalifa	Dubai	VAR	830
Shanghai Tower	Shanghai	China	632
Abraj Al Bait	Mecca	Saudi Arabia	601
Ping An Finance Center	Shenzen	China	599
Goldin Finance	Tainjin	China	597
Lotte World Tower	Seoul	South Korea	555
1 WTC	New York	USA	541
Guangzhou CTF Finance Center	Guangzhou	China	530
China Sun Tower	Beijing	China	528
Taipei 101	Taipei	Taiwan	508
World Finance Center	Shanghai	China	492
Lakhta Center	Saint Petersburg	Russia	462
Vincom Landmark 81	Ho Chi Minh City	Vietnam	461
Petronas Towers	Kuala Lumpur	Malaysia	452
Berlin TV Tower	Berlin	GDR	368

Program 15.3: Represent each tower by a column

```
towers.tab
TOWER::=TOWER subtext 1!12 # By this shortening of the name are also
                           # in the bar chart all names at the same time
                           # visible; for space reasons, otherwise
```

```
# sometimes some hidden
```

Result (diagram columns)



Program 15.4: Represent each tower by a bar and output the bars country by country. Countries with the highest towers are to be output first (sort downwards). For each country, sort the towers upwards. The countries are to be visually marked off.

```
aus towers.tab
gib MAX,COUNTRY,(HEIGHT,CITY m) m- MAX:=HEIGHT!max
COUNTRY::=COUNTRY wort + "-----------------------------------" subtext 1!30
RGBDARKGREEN:=darkgreen leftat MAX
RGBGREEN:=green leftat HEIGHT
```

upper part of the result (diagram bar)

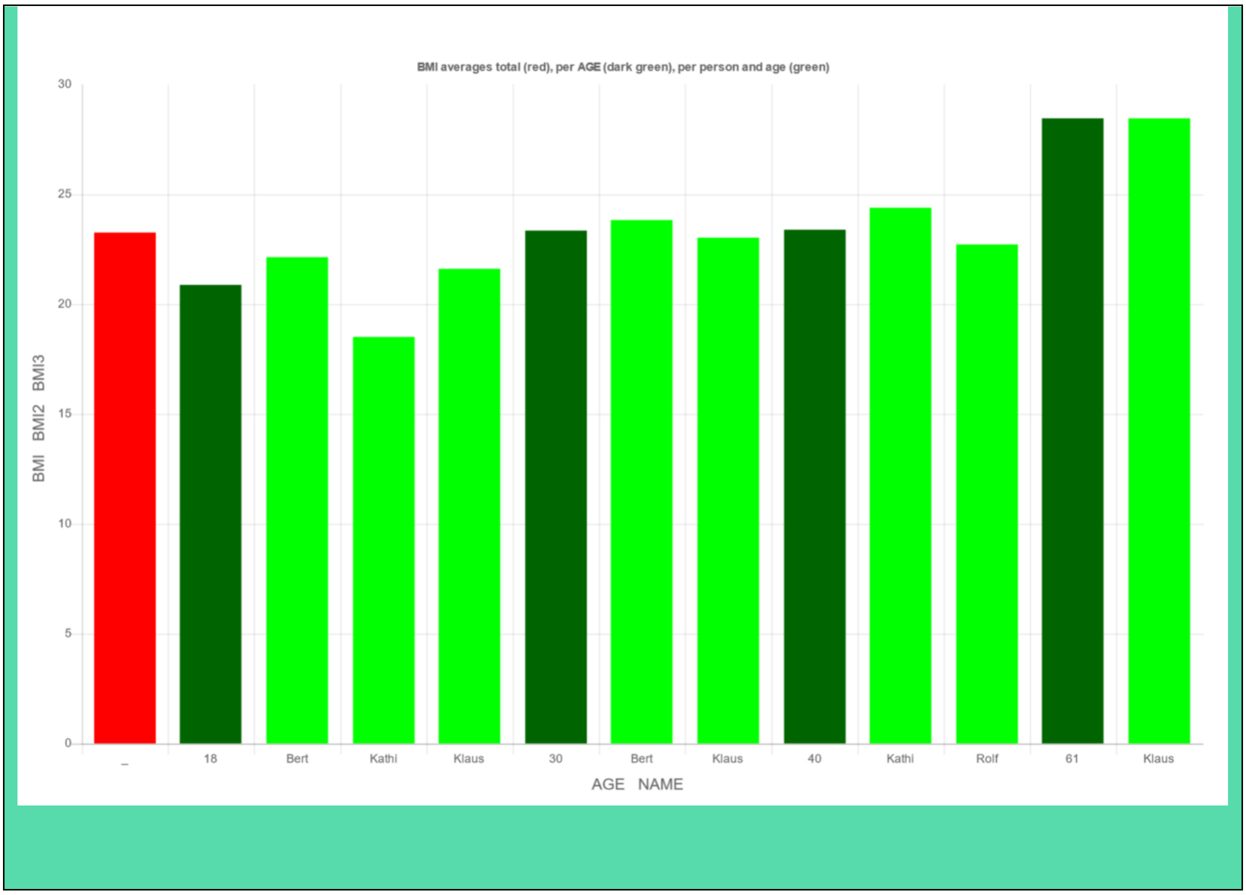
RGBDARKGREEN,MAX,COUNTRY,(RGBGREEN,HEIGHT,CITY m) m-
VAR
Dubai
China
Shanghai
Beijing
Guangzhou
Tainjin
Shenzen
Shanghai
Saudi_Arabia
Mecca
South_Korea
Seoul
USA
New_York
Taiwan
Taipei
Russia
Saint_Petersburg
Vietnam
Ho_Chi_Minh_City
Malaysia
Kuala_Lumpur
GDR
Berlin
COUNTRY CITY
0
100
200
300
400
500
600
700
800
900
MAX HEIGHT

```
weights.tabh
NAME,    LENGTH, (AGE,WEIGHTl l) l
Bert     1.72     18  66 65
                  30  70 71
Kathi    1.7      18  55 52
                  40  70 71
Klaus    1.68     18  61 60 62
                  30  65 63 67
                  61  80 82 79
Rolf     1.78     40  72 70 74
Victoria 1.61     13  51 50
Walleri  1.       3   16 15
```

Program 15.5: Calculate BMI averages for all adults, for each age group, and overall. To realize the first 5 lines EXCEL needs more than 6 worksheets**.**

```
aus  weights.tabh
avec NAME! AGE>20
gib  BMI,(AGE,BMI,(NAME,BMI m) m)
     BMI:=WEIGHT:LENGTH:LENGTH ! ++:
rnd 2
AGE::=AGE wort
RGBRED:=red leftat BMI
RGBDARKGREEN:=darkgreen leftat BMI2
RGBGREEN:=green leftat BMI3
TITEL:="BMI averages total (red), per AGE (dark green), "
       + "per person and age (green)" leftat RGBRED
```

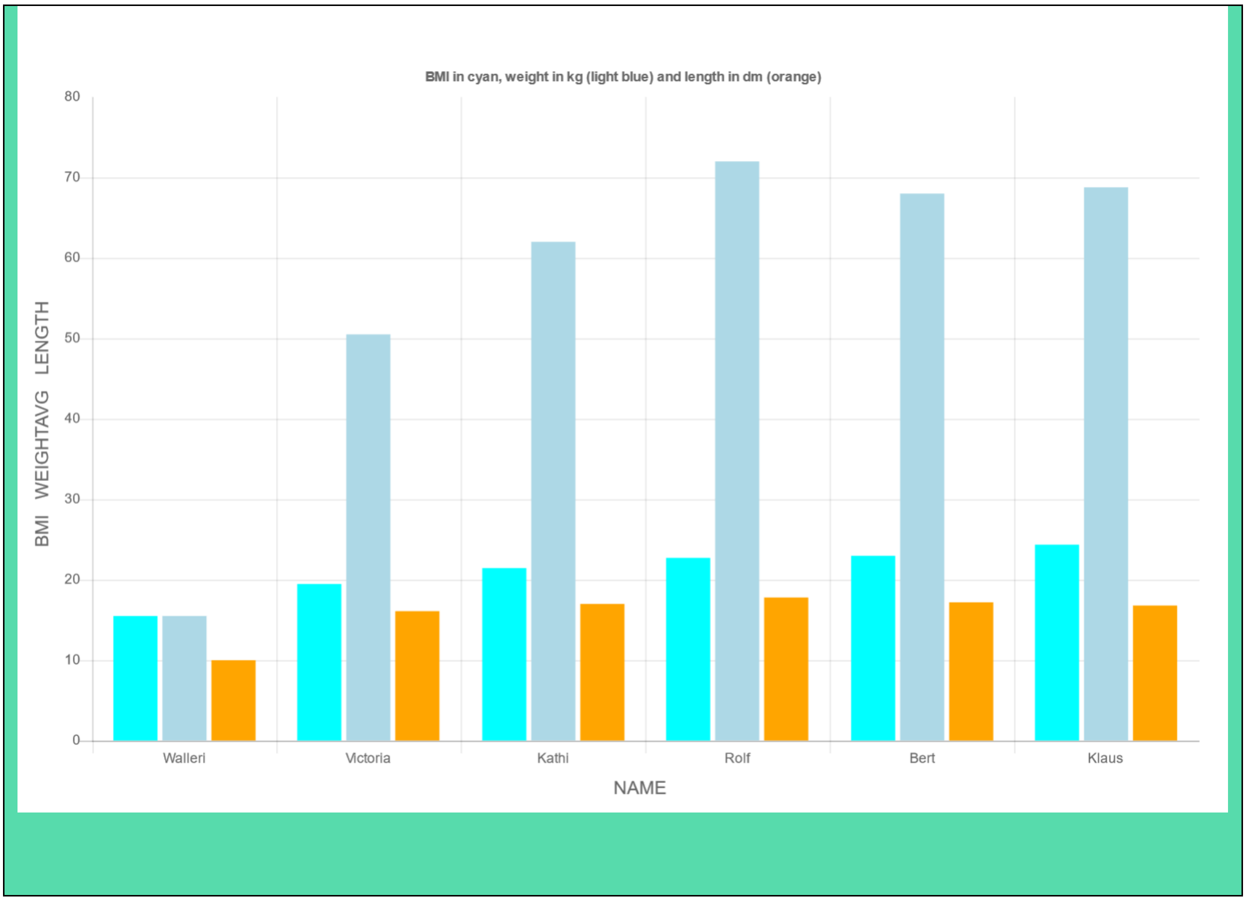
```
Result (diagram Säulen (columns))
```



Program 15.6: Compare the weights, lengths and BMI of all persons. Sort the persons by BMI.

```
aus weights.tabh
TITEL:="BMI in cyan, weight in kg (light blue) and length in dm (orange)"
gib TITEL,(BMI,NAME,WEIGHTAVG,LENGTH m)
    WEIGHTAVG:=WEIGHT!++:
    BMI:=WEIGHT:LENGTH:LENGTH ! ++:
LENGTH::=LENGTH*10
AGE::=AGE wort
RGBLIGHTBLUE:=lightblue leftat WEIGHTAVG
RGBORANGE   :=orange    leftat LENGTH
RGBCYAN     :=cyan      leftat BMI
```

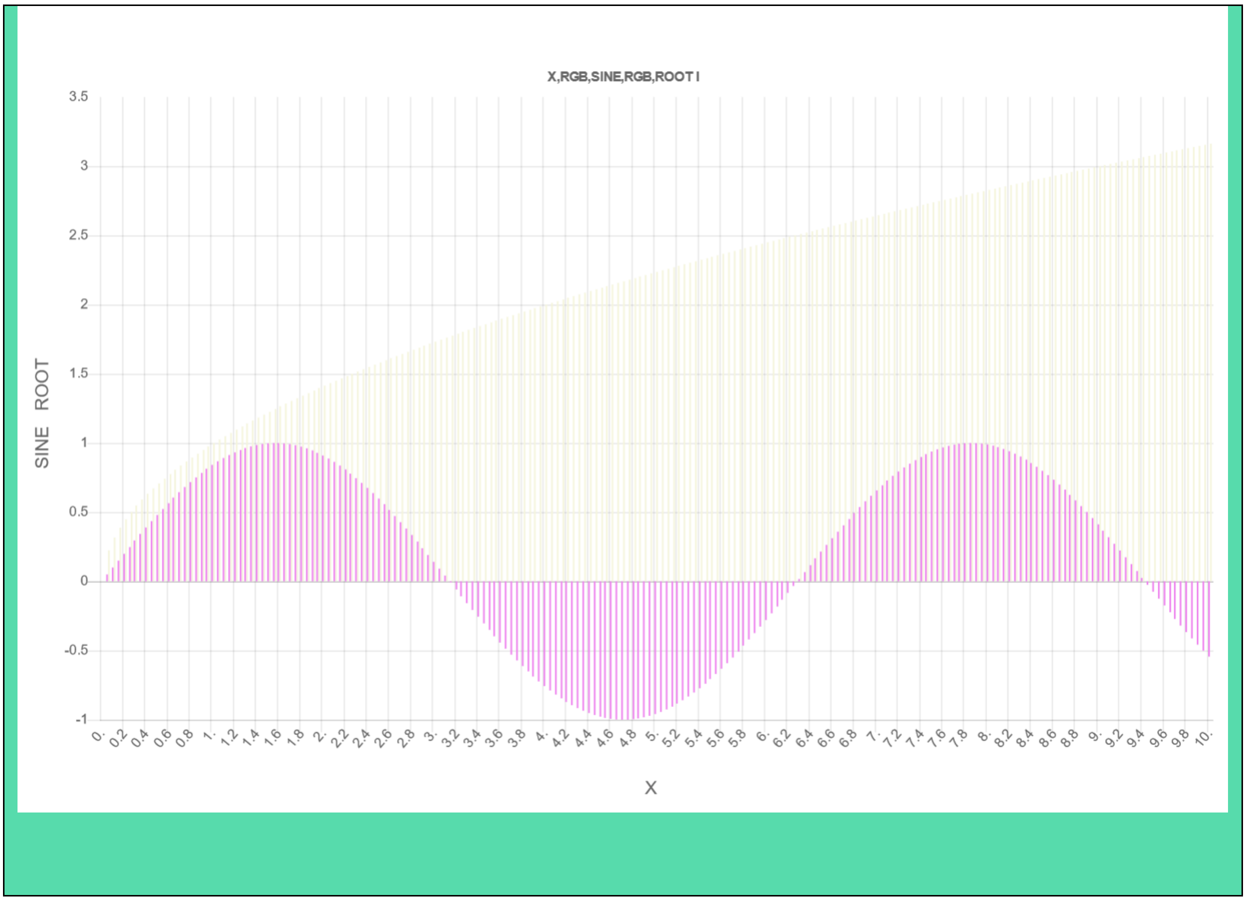
Result (diagram columns)



Program 15.7: Represent 2 functions by bar graphs.

`Xl:=0 ...10!0.05`
`SINE :=X sin`
`ROOT:=X sqrt`
`X::=X wort`
`RGB:=violet leftat SINE`
`RGB:=beige  leftat ROOT`

Result (diagram columns)

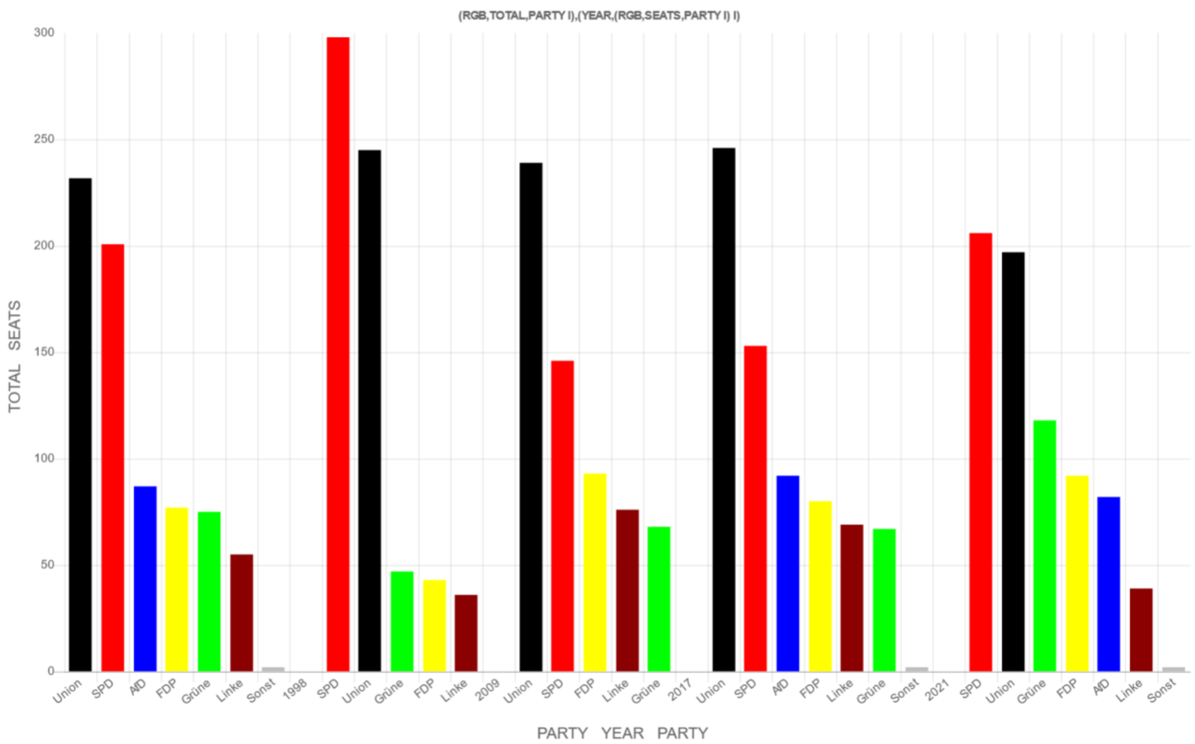



```
elections.tab
YEAR, (PARTY,SEATS l) l
1998    PDS     36
        SPD    298
        Grüne   47
        Union  245
        FDP     43
2009    Linke   76
        SPD    146
        Grüne   68
        Union  239
        FDP     93
2017    Linke   69
        SPD    153
        Grüne   67
        Union  246
        FDP     80
        AfD     92
        Sonst    2
2021    Linke   39
        SPD    206
        Grüne  118
        FDP     92
        Union  197
        AfD     82
        Sonst    2
```

Program 15.8: Visualize 4 election results and calculate the average number of votes of the 4 elections.

```
elections.tab
YEAR::=YEAR wort
PARTY::=Linke if PARTY="PDS" ! PARTY
gibl TOTAL,PARTY m-,(YEAR,(SEATS,PARTY m-)m) TOTAL:=SEATS! ++:
RGB:=red     if PARTY="SPD" !
     yellow  if PARTY="FDP" !
     darkred if PARTY=Linke !
     blue    if PARTY=AfD   !
     black   if PARTY=Union !
     green   if PARTY=Grüne !
     grey    leftat TOTAL SEATS
```

Result (diagram columns)




```
Result without RGB values (tab)
TOTAL ,PARTY  l,(YEAR ,(SEATS ,PARTY  l) l)
231.75 Union     1998   298    SPD
200.75 SPD              245    Union
 87.   AfD               47    Grüne
 77.   FDP               43    FDP
 75.   Grüne             36    Linke
 55.   Linke     2009   239    Union
  2.   Sonst            146    SPD
                         93    FDP
                         76    Linke
                         68    Grüne
                 2017   246    Union
                        153    SPD
                         92    AfD
```

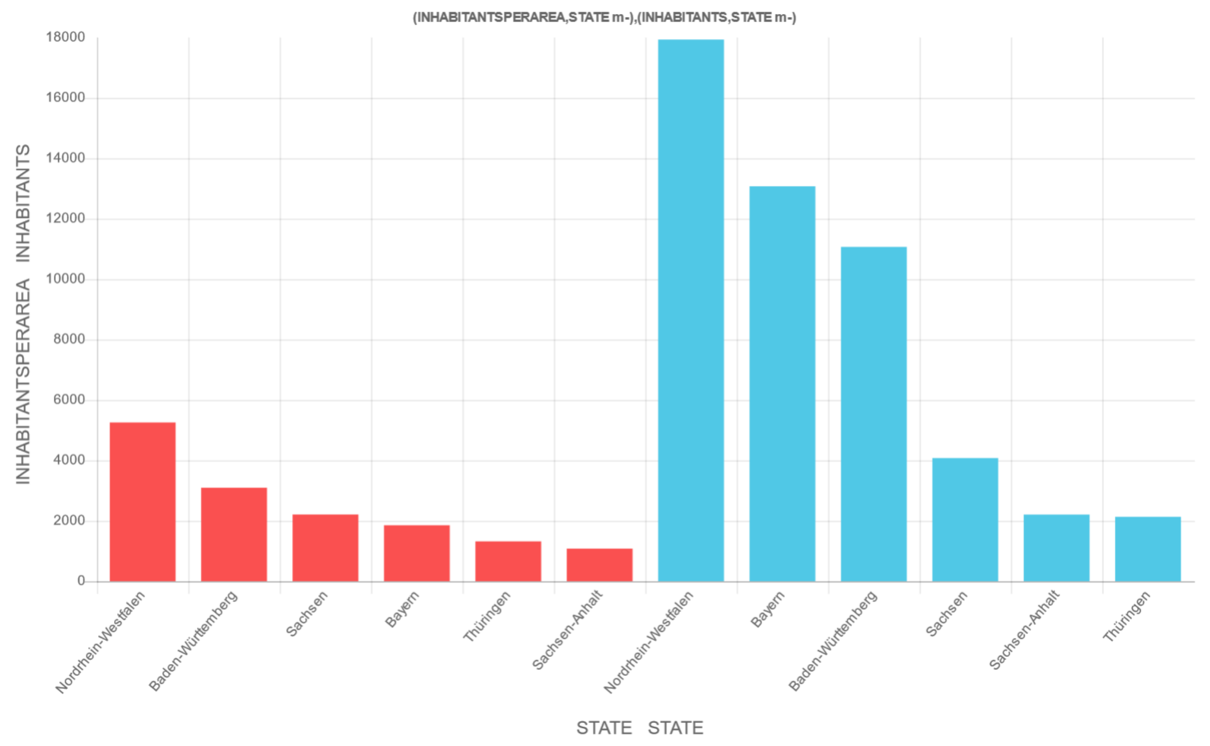
```
                80    FDP
                69    Linke
                67    Grüne
                 2    Sonst
        2021   206    SPD
               197    Union
               118    Grüne
                92    FDP
                82    AfD
                39    Linke
                 2    Sonst
```

```
states6.hsq
STATE, SHORT, AREA, INHABITANTS l
Sachsen             SN  18'449.99  4'077
Sachsen-Anhalt      ST  20'451.58  2'208
Thüringen           TH  16'172.5   2'134
Bayern              BY  70'541.57 13'076
Baden-Württemberg   BW  35'751.46 11'069
Nordrhein-Westfalen NRW 34'110.26 17'932
```

Program 15.9: Sort and visualize the states by population per area and by population. (Visualize 2 independent tables.)

```
states6.hsq
INHABITANTSPERAREA:=INHABITANTS : AREA *10'000
gib INHABITANTSPERAREA,STATE m- , (INHABITANTS,STATE m-)
'3
rnd 0
```

```
Result (diagram columns)
```

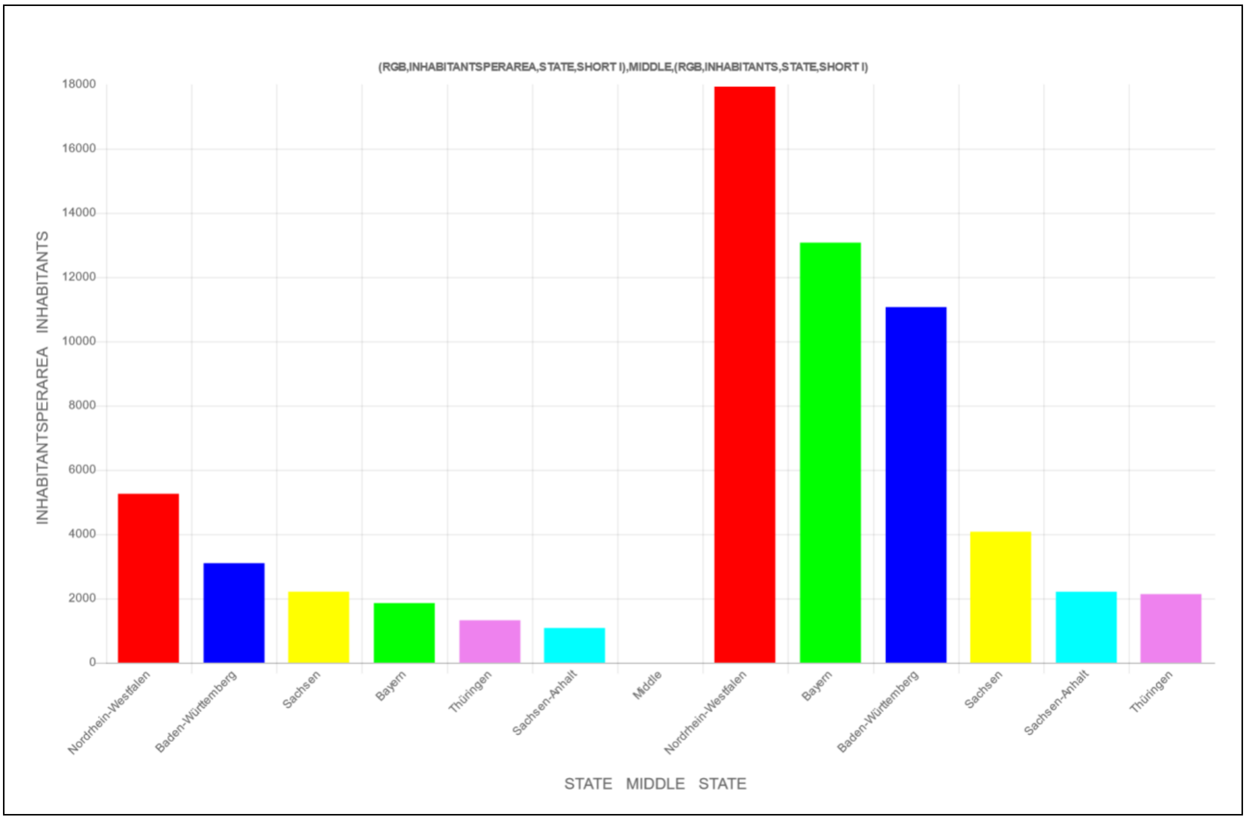



```
Result (rounded to 0 digits and '3)
INHABITANTSPERAREA ,STATE              m-,(INHABITANTS ,STATE              m-)
5'257.              Nordrhein-Westfalen 17'932       Nordrhein-Westfalen
3'096.              Baden-Württemberg   13'076       Bayern
2'210.              Sachsen             11'069       Baden-Württemberg
1'854.              Bayern               4'077       Sachsen
1'320.              Thüringen            2'208       Sachsen-Anhalt
1'080.              Sachsen-Anhalt       2'134       Thüringen
```

Program 15.10: Sort and visualize the states by population per area and by population, such that each states gets the same color in each of the diagrams. Divide the tables by additional space.

```
states6.hsq
INHABITANTSPERAREA:=INHABITANTS : AREA *10'000
MIDDLE:=Middle
gibl INHABITANTSPERAREA,STATE,SHORT m- ,MIDDLE,(INHABITANTS,STATE,SHORT m-)
RGB:= red    if SHORT="NRW" !
      blue   if SHORT="BW"  !
      yellow if SHORT="SN"  !
      green  if SHORT="BY"  !
      violet if SHORT="TH"  !
      cyan   leftat INHABITANTSPERAREA INHABITANTS
```

```
Result (diagram columns)
```


(RGB,INHABITANTSPERAREA,STATE,SHORT I),MIDDLE,(RGB,INHABITANTS,STATE,SHORT I)
INHABITANTSPERAREA INHABITANTS
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
Nordrhein-Westfalen
Baden-Württemberg
Sachsen
Bayern
Thüringen
Sachsen-Anhalt
Middle
Nordrhein-Westfalen
Bayern
Baden-Württemberg
Sachsen
Sachsen-Anhalt
Thüringen
STATE MIDDLE STATE

16 Multiple diagrams

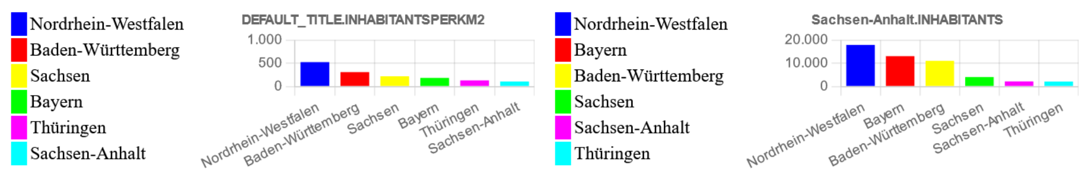
In the previous chapter we saw that a structured table can usually also be represented as a structured chart. Program 15.6 demonstrates that this also works for larger tables. However, pie charts quickly become confusing if a circle represents too many numbers.

Structured tables usually contain several sub-tables. These naturally contain fewer elements than the source table, so in this chapter each sub-table will be represented by a diagram. With multiple diagrams, structured tables are visualized even more directly than with structured diagrams.

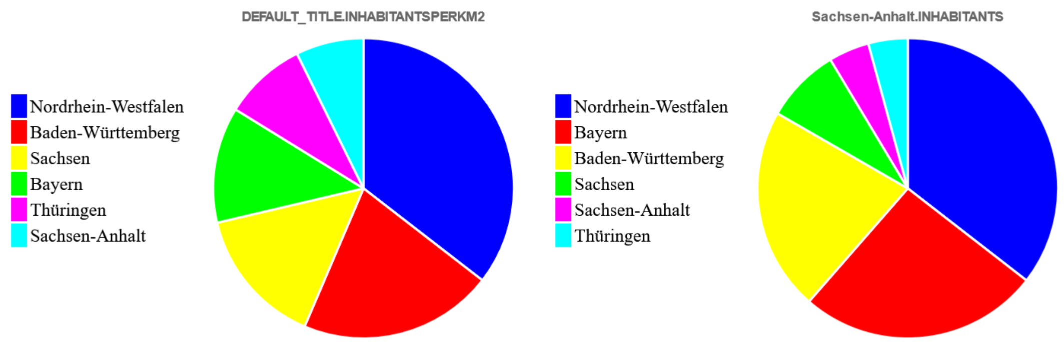
Program 16.1: Sort and visualize the states by population per square kilometer and by population. (Visualize 2 independent tables.)

```
states6.hsq
INHABITANTSPERKM2:=INHABITANTS : AREA *1'000
gib INHABITANTSPERKM2,STATE m- , (INHABITANTS,STATE m-)
'3
rnd 0
```

Result (2 bar charts)



Result (2 pie charts)

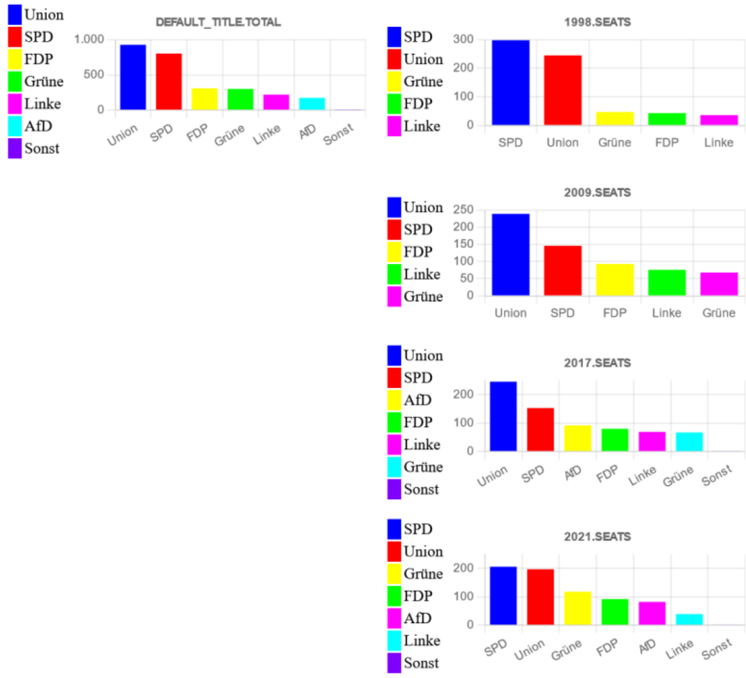


In program 15.9, the inhabitants per square kilometer were multiplied by 100,000 to make the columns clearly visible. This adjustment is not necessary for multiple charts.

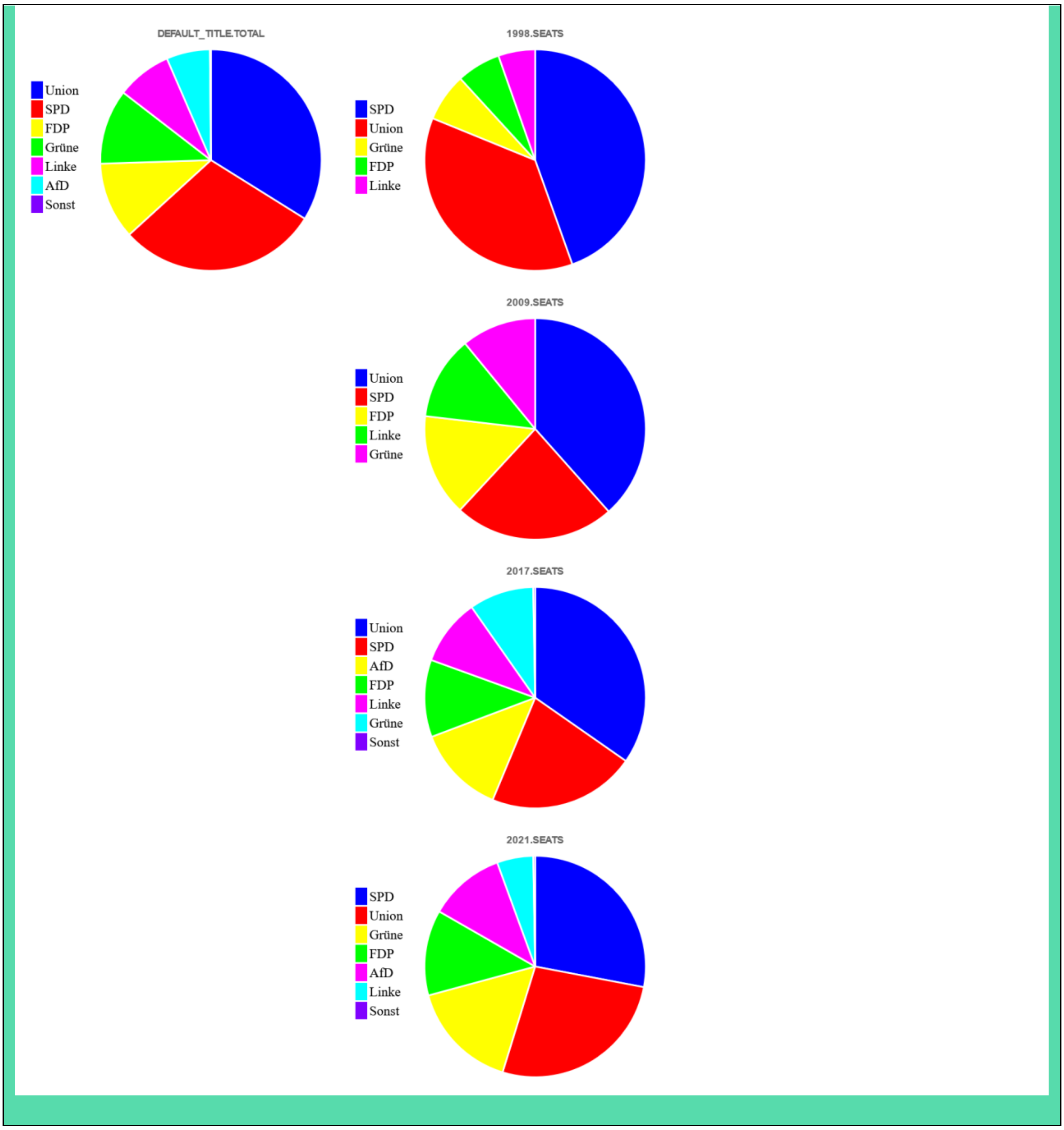
Program 16.2: as 15.8

```
elections.tab
YEAR::=YEAR wort
PARTY::=Linke if PARTY="PDS" ! PARTY
gib TOTAL,PARTY m-,(YEAR,(SEATS,PARTY m-)m) TOTAL:=SEATS! ++
```

Result (5 bar charts)



Result (5 pie charts)



Note the difference between the above program and program 15.8. In 15.8 the sum of the four files is calculated and here the average is calculated. Therefore, the order of the parties in the total balance differs. The order does not change even if an "average" is calculated by division by 4. Here you can see how important it is that the end user must be able to read the program in order to correctly process the information received.

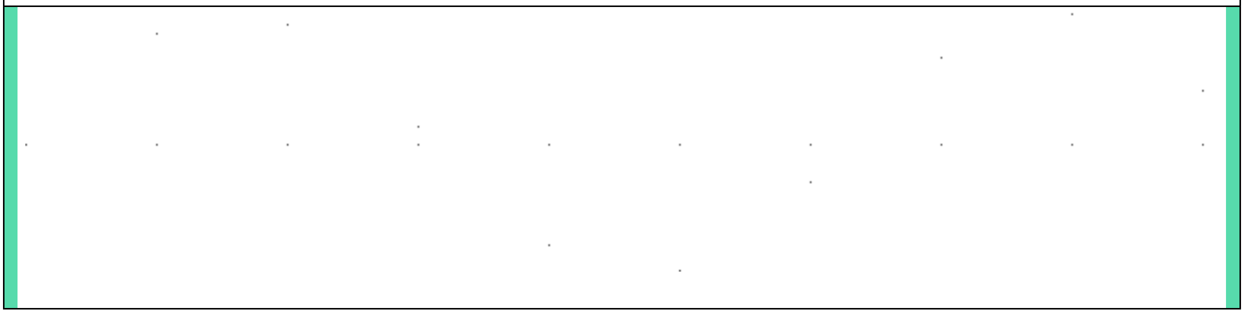
17 Image generation

Since with o++o numbers can be generated in a simple way, one can also generate whole images. For example, Xl:= 0 .. 4 generates the numbers 0 1 2 3 4. You can assign diagrams to these numbers, but to generate an image with o++o you need a list or a set of number pairs (X,Y). The point gets a color if there is a RGB (RED,GREE,BLUE) triple before the X or before the Y value:
(X, RGB, Y)
For RGB values o++o has 3 display options.
English color names
red, silver, cyan, ...
Triples of integers between 0 and 255
(255,0,0) (=red), (192,192,192) (=silver), (0,255,255) (=cyan)
Number triples between 0 and 1:
(1.,0.,0.) (=rot),(0.752941,0.752941;0.752941)(=silver),(0.,1.,1.) (=cyan)
We start with 2 functions, but initially define them only for 10 X values. You have to look closely to see the points:

Program 17.1: Create 10 points twice.

```
Xl:=0 ..9
Y :=X sin
Y0:=X*0
```

```
Result (image - new window)
```

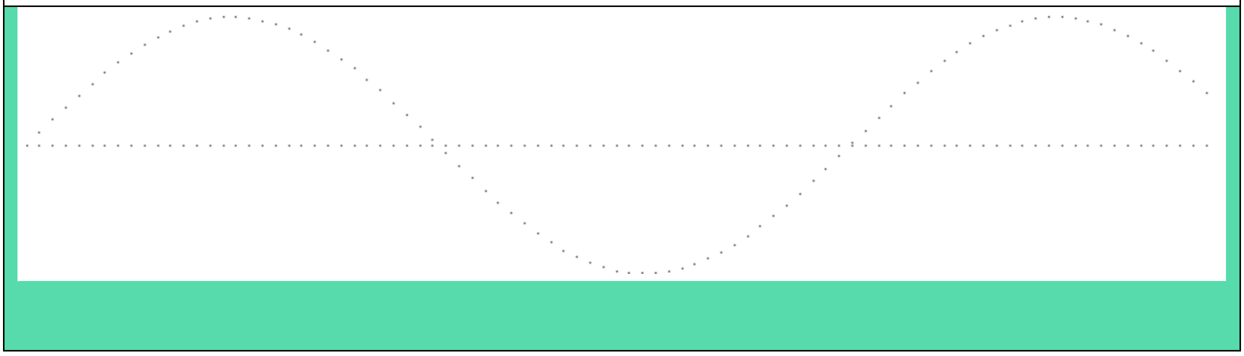


By introducing a step size of 0.1, the number of points is increased tenfold.

Program 17.2: Create 100 points twice.

```
Xl:= 0 ...9!0.1
Y:=X sin
Y0:=X*0
```

```
Result (image - new window)
```

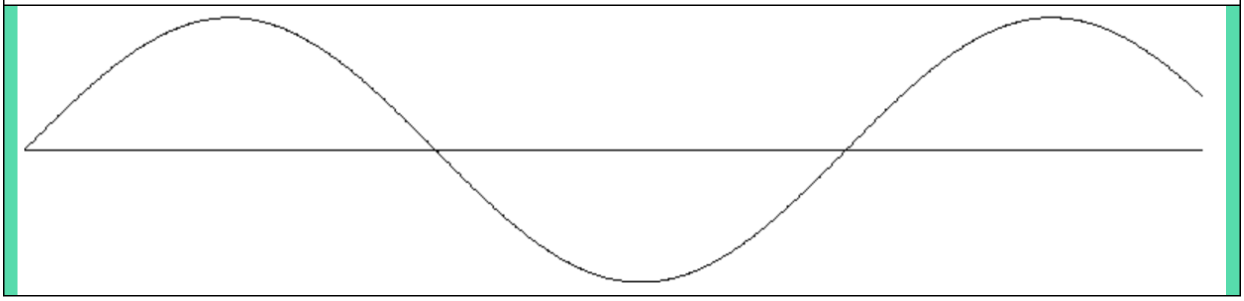


Now we add another 0 to the step size.

Program 17.3: Create 1000 points twice.

```
Xl:=0 ...9!0.01
Y :=X sin
Y0:=X*0
```

```
Result (image - new window)
```

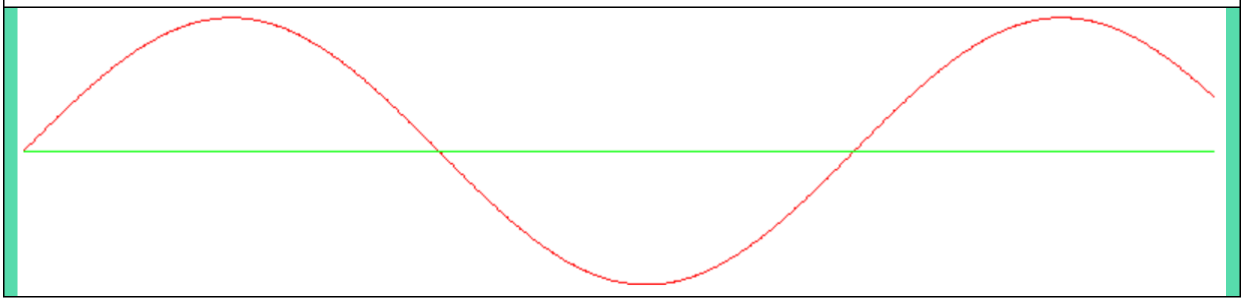


The sine function now becomes red and the X-axis green. The fact that a column name occurs twice (RGB) does not cause any problems at this point.

Program 17.4: Display 2 functions in color.

```
Xl:= 0 ...9!0.01
Y :=X sin
Y0:=X*0
RGB:=red leftat Y
RGB:=green leftat Y0
```

```
Result (image - new window)
```



The fact that it is also possible to create "full images" is first shown by the German flag. You can see that all points that follow a color value are output in this color. Thus, only three color values are needed for the German flag. The term pixel has lost its meaning here or must be redefined.

Program 17.5: Generate the German flag

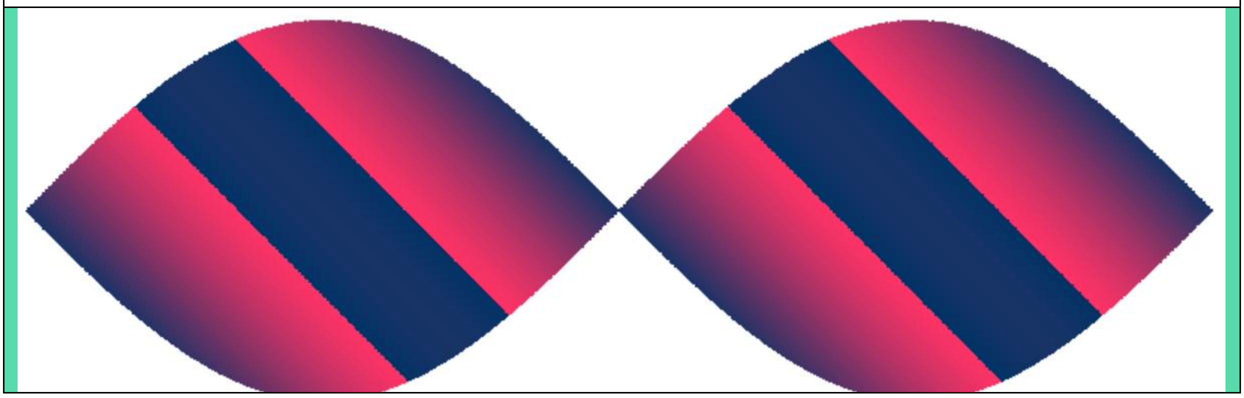
```
Xl:= 0 ...9!0.01
Yl:= 0 ...2!0.01 at X
=: $RECTANGLE
aus RGB:=gold
,$RECTANGLE
RGB:=red
,$RECTANGLE+(0,2)
RGB:=black
,$RECTANGLE+(0,4)
```

Result (image + new window)

Program 17.6: Design a bikini. Color the functions mirrored between sine and sine mirrored.

```
Xl:=pi * -1 ...pi!0.005
Yl:=X sin abs *-1 ...(X sin abs)! 0.005
RGB:= 0.1+(X+Y sin abs),0.2,0.4 leftat Y
```

Result (image + new window)



18 Image editing

The following programs are all based on the photo of an Indian:

Program 18.1: Display a photo!
inder.jpg
`Result (image)`
`(see above)`

Above image takes only 42.2 KByte.

In the following it becomes clear that the file contains 160 thousand records (tuples).

Program 18.2: Count the lines (pixels) of the photo.

```
inder.jpg ++1 '3
```

Result (tab)

```
160'001
```

Usually, it is not possible to look at all points of an image. Therefore, we only output the first 14 5-tuples in the following.

Program 18.3: Select the first 14 image pixels.

```
inder.jpg
avec X pos < 15
```

Result (tab)

```
X, RGB, Y l
0. 228,232,220 0.
0. 227,231,219 0.01
0. 227,231,219 0.02
0. 226,230,218 0.03
0. 227,231,219 0.04
0. 228,232,220 0.05
0. 229,233,221 0.06
0. 230,234,222 0.07
0. 229,233,221 0.08
0. 229,234,221 0.09
0. 229,234,221 0.1
0. 229,234,221 0.11
0. 229,233,221 0.12
0. 229,233,220 0.13
```

Program 18.4: Calculate the extreme values of the coordinates of the image points

```
inder.jpg
gib XMAX,XMIN,YMAX,YMIN
    XMAX:=X!max XMIN:=X!min
    YMAX:=Y!max YMIN:=Y!min
```

Result (tab)

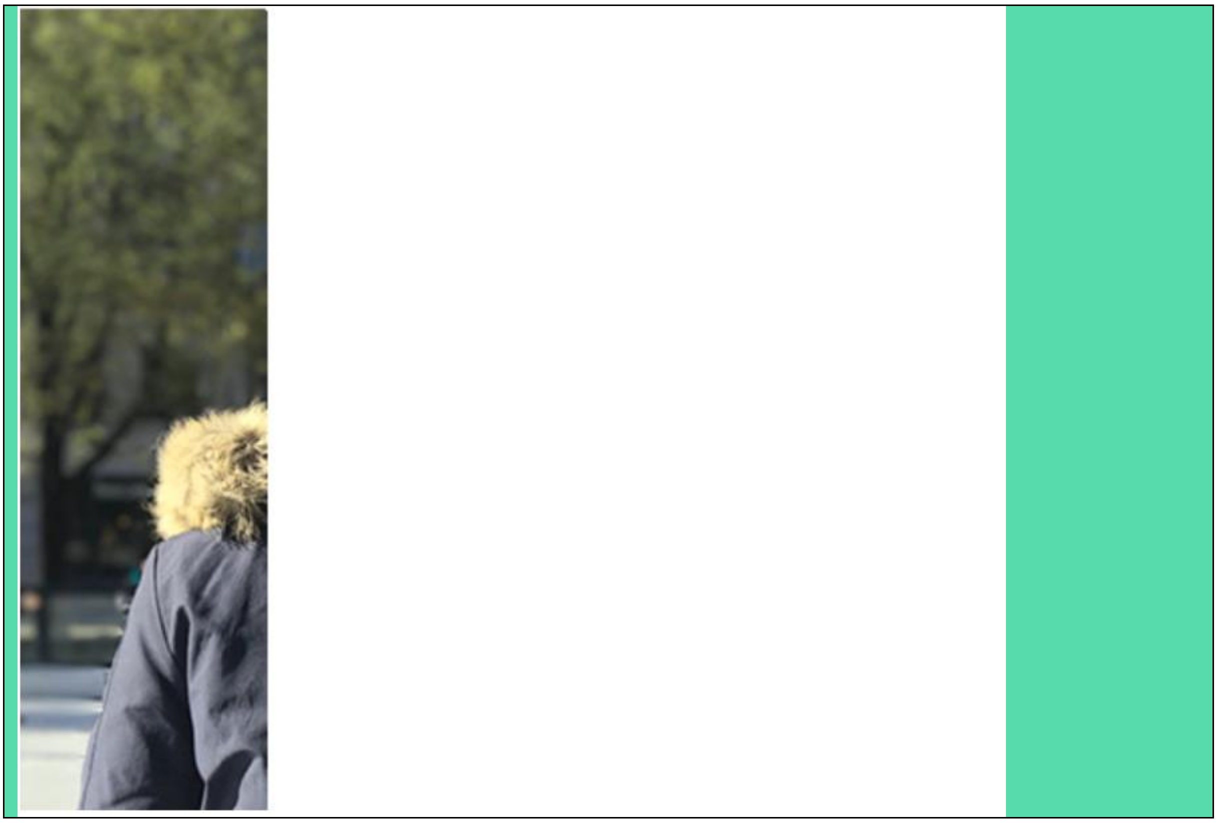
```
XMAX, XMIN, YMAX, YMIN
4.02  0.    3.99  -0.05
```

You can see here that the photo represents a 4 x 4 square. The X and Y values are therefore between 0 and 4.

Program 18.5: Output the left part of the photo.

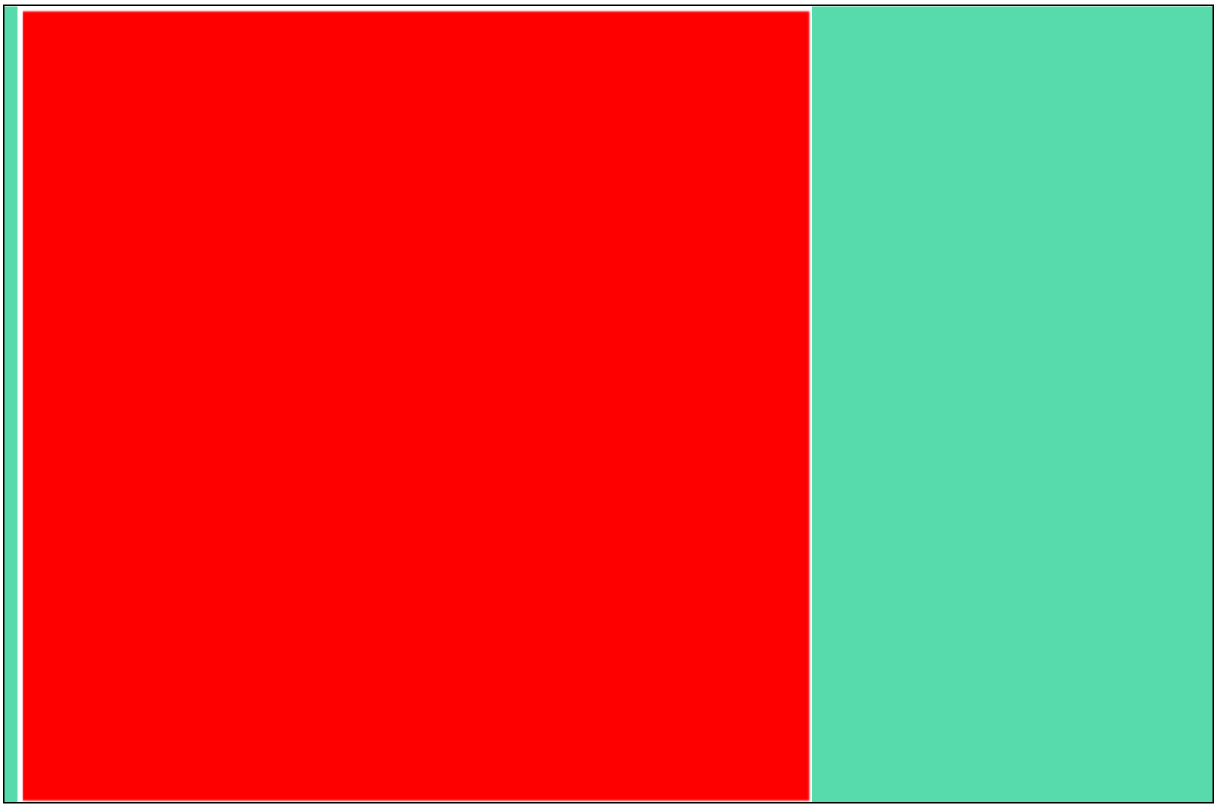
```
inder.jpg
avec X pos < 50'000
```

Result (image)



A simple red rectangle can also be created with any photo by simply overwriting any color value.

Program 18.6: Paint the photo red.
`inder.jpg` `RGB::= red`
`Result (image-new window)`

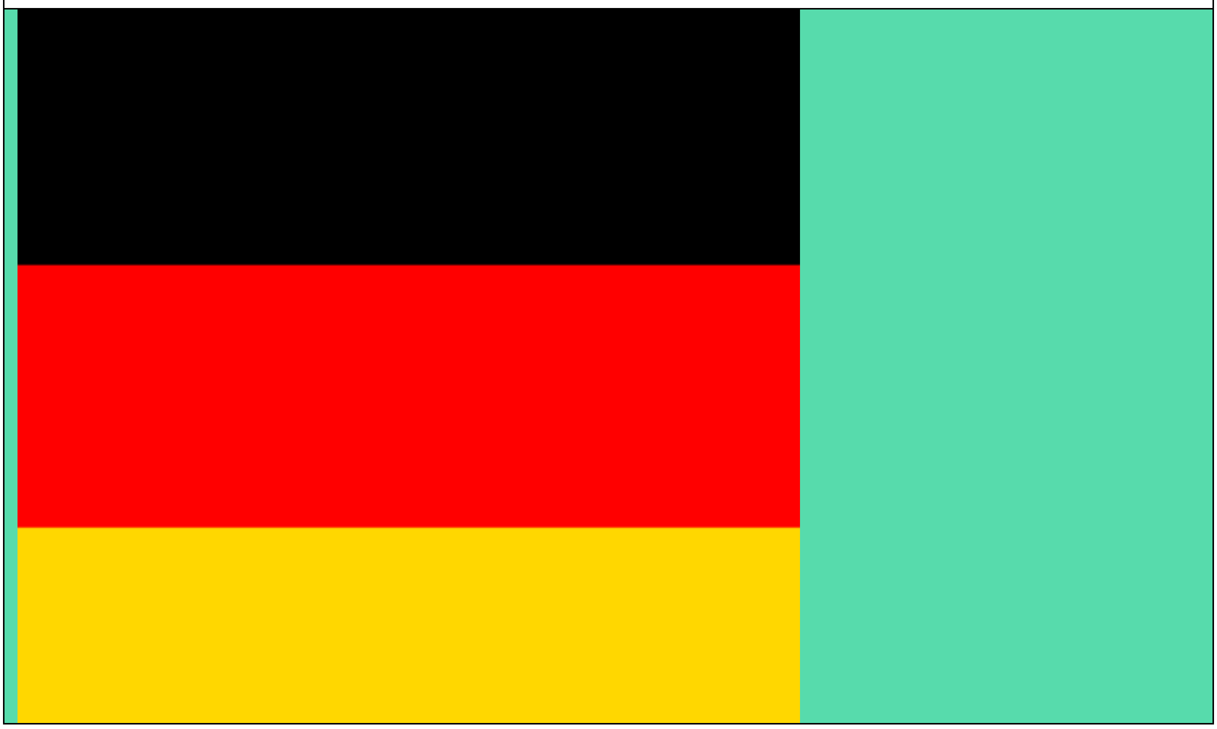


Also, very easy to create the German flag from the photo:

Program 18.7: Transform a photo to the German flag.

```
inder.jpg
RGB::= black if Y>8:3 !
       red   if Y>8:6 !
       gold
```

Result (image - new window)



Program 18.8: Hide the head behind a brown bar.

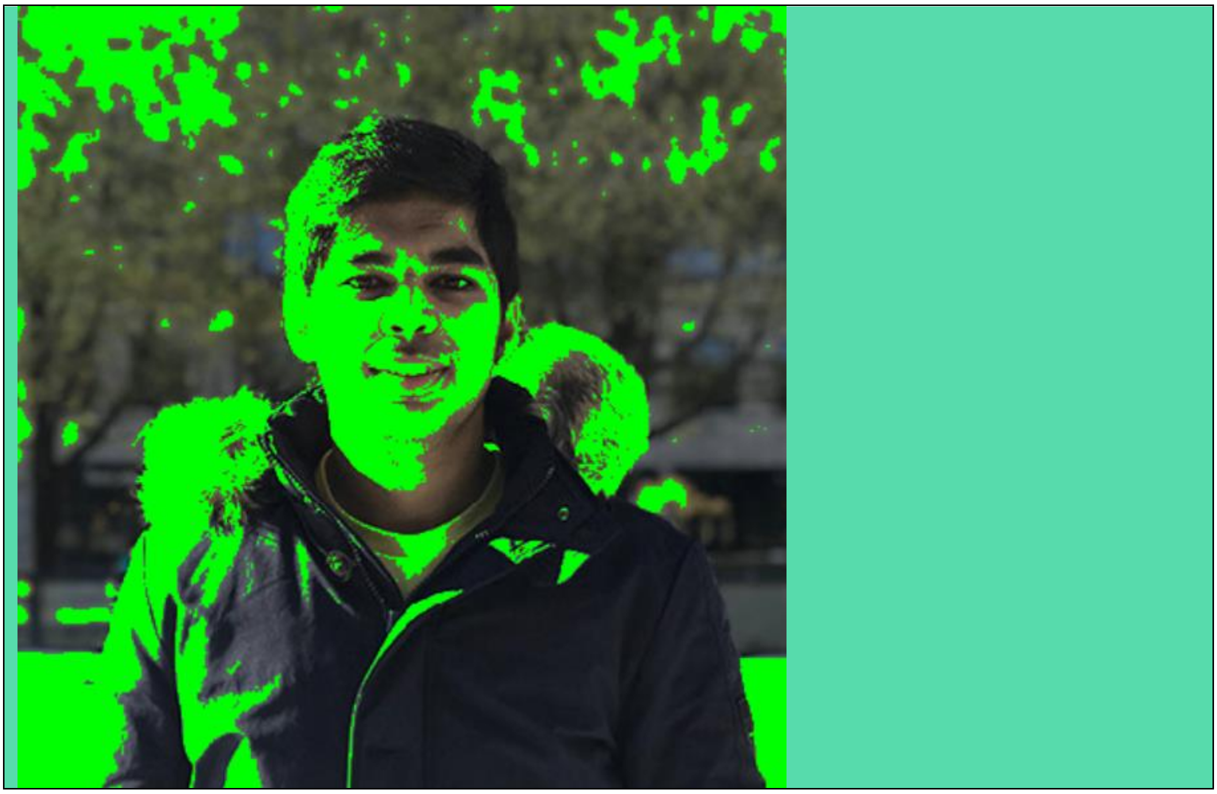
```
inder.jpg
RGB::= brown if 1.5<X & X<2.5 ! RGB
```

Result (image - new window)

Program 18.9: Convert any stronger green to pure green.

```
inder.jpg
RGB::= green if RGB nth 2>100!RGB
```

Result (image - new window)



Program 18.10: Convert a color photo into a black-cyan photo.

```
inder.jpg
RGB::= cyan if RGB ++ <220!black
```

Result (image - new window)

Program 18.11: Output the photo including a displacement of the photo.

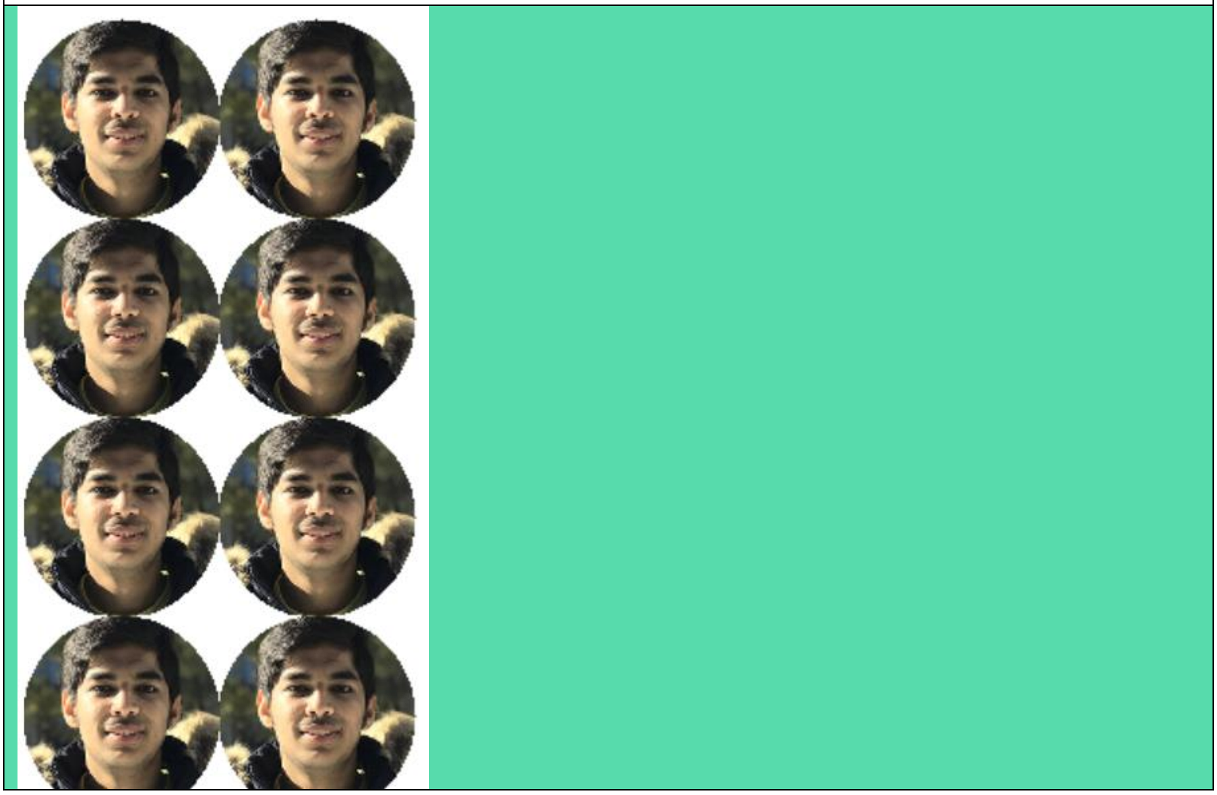
```
inder.jpg
Y:=inder.jpg+(8.,0,0,0.)
```

Result (image - new window)

Program 18.12: Arrange the head of the Indian in a 2*4 rectangle.

```
inder.jpg
avec X- 2 ^ 2 +(Y - 2.3 ^ 2) <1
=: $HEAD
aus Zl:= 1 ..4
W1:= $HEAD + (0.;0;0;2*Z)
W2:= $HEAD + (2.;0;0;2*Z)
```

Result (image - new window)



In the following it becomes clear that the Indian is still clearly visible, although we have only taken over its red part. A blue wave starts at the top right of the following image, regardless of the given colors.

Program 18.13: Create a blue wave over the red parts of the photo.
`inder.jpg` `RGB::=RGB nth 1;0 ;X+Y sin abs`
`Result (image)`

Even though we have only given a few examples here, it should have become clear that with o++o you can create thousands or millions of different images from a single photo.

19 A baker application (CSS)

In this chapter, form questions take up somewhat more space than the content questions of the otto- program. The baker wants to create invoices and then be able to evaluate the collected invoice data. The following program generates the invoice and saves the essential data of all invoices immediately into a file rechnungen22.hsq, which can be re-evaluated immediately after an invoice is created.

Program 19.1 Printing invoices for customers and saving them to a file.

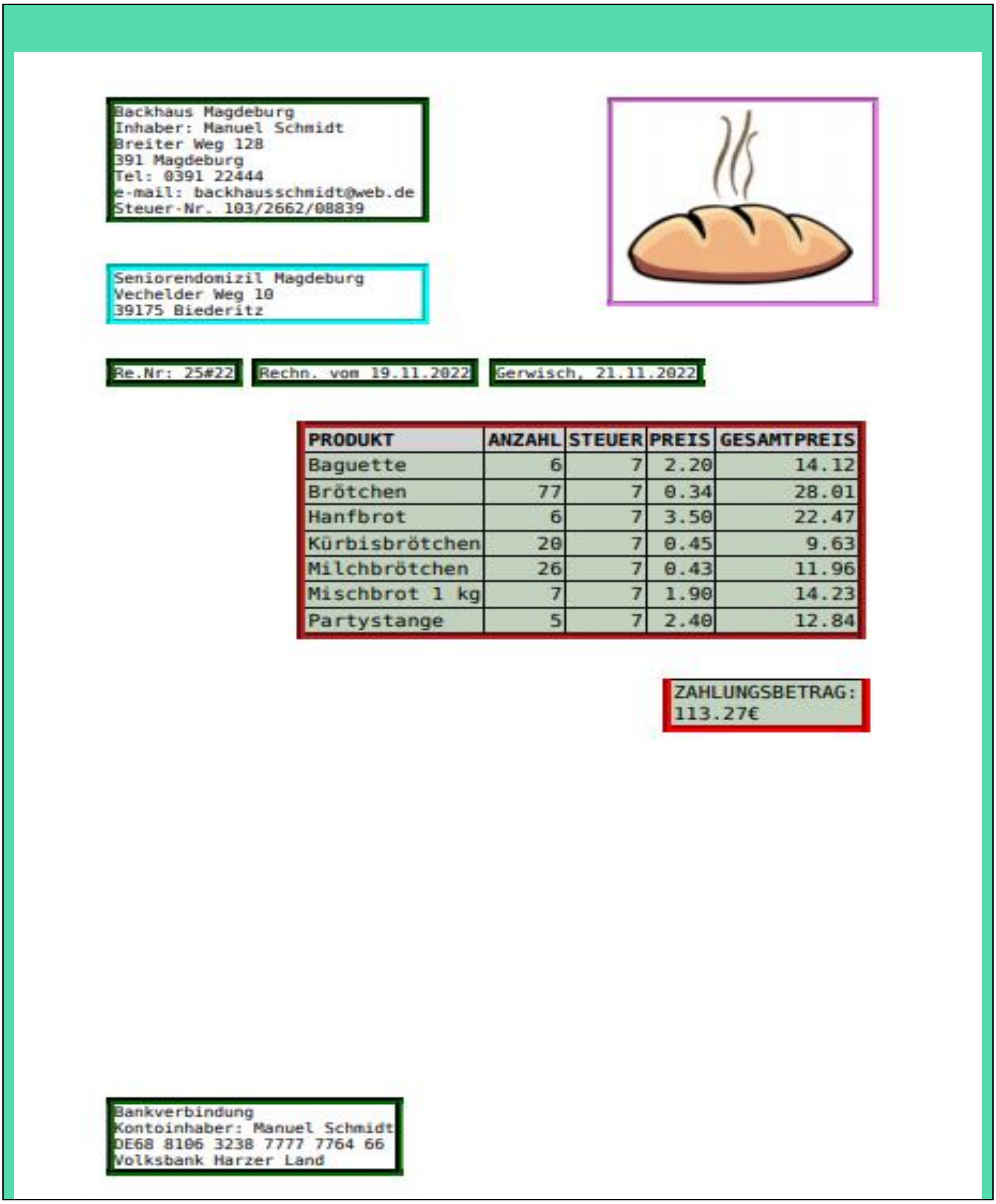
```
$RECHNNR :="25#22"
$KUNDE :="Senioren"
$RECHNVOM :="19.11.2022"
$DATUM :="21.11.2022"
$BEZAHLT :=nein
$BESTELLUNG:= <TAB!
PRODUKT,       ANZAHL m
Hanfbrot        6
Baguette        6
Brötchen       77
Kürbisbrötchen 20
Partystange     5
Mischbrot 1 kg  7
Milchbrötchen  26
!TAB>
################## ENDE DER EINGABE #########################
$FALSCHEPRODUKTE:=begin aus $BESTELLUNG;; gib PRODUKTm
-coll begin aus produkte.tab;;gib PRODUKTm end end
$KUNDEANZ:=begin aus kunden.tab; avec KUNDE = $KUNDE; ++1 end

aus ($KUNDEANZ = 0 | $FALSCHEPRODUKTE ++1>0) dann (($KUNDEANZ=0) dann
"Kunde inkorrekt" ! ("falsche Produkte:" ; $FALSCHEPRODUKTE)) !
    begin
aus $BESTELLUNG,produkte.tab
igib (PRODUKT,STEUER,PREIS,ANZAHL) m
=: $BESTELLUNGGESAMT
aus $BESTELLUNGGESAMT
,absender.tab
,begin aus kunden.tab;avec KUNDE = $KUNDE;gib KUNDENADRESSE
KUNDENADRESSE=NAME,STRASSE,ORT end
GESAMTPREIS := STEUER :100 +1 *PREIS *ANZAHL
gib ABSENDER,KUNDENADRESSE,TABELLEN
    TABELLEN= ZEILEN,ZAHLUNGSBETRAG
    ZEILEN= PRODUKT,ANZAHL,STEUER,PREIS,GESAMTPREIS l
    ZAHLUNGSBETRAG:=GESAMTPREIS!++
rnd 2
,bankverbindung.tab
BILD:= "bilder/brotraucht.png" at ABSENDER
$L:="Re.Nr: " + ($RECHNNR text)
$M:="Rechn. vom " + ($RECHNVOM text)
$R:="Gerwisch, " + ($DATUM text )
BETREFF:= $L,$M,$R leftat TABELLEN
ZAHLUNGSBETRAG::="ZAHLUNGSBETRAG: " + (ZAHLUNGSBETRAG text) + " €"
tag0 RECHNUNG
FORMAT:=format2.tab
=: $ERGEBNIS
```

```
aus $RECHNNR,$DATUM,$KUNDE,$BEZAHLT,$BESTELLUNGGESAMT
gib RECHNNR,DATUM,KUNDE,BEZAHLT,(PRODUKT,ANZAHL,STEUER,PREIS m) m
,rechnungen.hsq
gib RECHNNR,DATUM,KUNDE,BEZAHLT,(PRODUKT,ANZAHL,STEUER,PREIS m) m
rnd 2
save rechnungen22.hsq

aus $ERGEBNIS
end
```

Result (web) (new window)



Backhaus Magdeburg
Inhaber: Manuel Schmidt
Breiter Weg 128
391 Magdeburg
Tel: 0391 22444
e-mail: backhausschmidt@web.de
Steuer-Nr. 103/2662/08839

Seniorendomizil Magdeburg
Vechelder Weg 10
39175 Biederitz

Re.Nr: 25#22 Rechn. vom 19.11.2022 Gerwisch, 21.11.2022

PRODUKT	ANZAHL	STEUER	PREIS	GESAMTPREIS
Baguette	6	7	2.20	14.12
Brötchen	77	7	0.34	28.01
Hanfbrot	6	7	3.50	22.47
Kürbisbrötchen	20	7	0.45	9.63
Milchbrötchen	26	7	0.43	11.96
Mischbrot 1 kg	7	7	1.90	14.23
Partystange	5	7	2.40	12.84

ZAHLUNGSBETRAG:
113.27€

Bankverbindung
Kontoinhaber: Manuel Schmidt
DE68 8106 3238 7777 7764 66
Volksbank Harzer Land

After the baker has entered his order data, it is checked whether all product names are also contained in the file products.tab. Then, all data of the order relevant for the costs are summarized in the tab variable $BESTELLUNGGESAMT. This variable has a schema of the type PRODUCT,TAX,PRICE,ZAHL m. I.e. it contains a whole table. To this data the customer data and the sender are added. The total price is calculated. The invoice requires additional tags so that the data can be formatted in the desired way. The whole invoice is cached in the $ERGEBNIS tab variable so that the invoice can be added to rechnungen22.hsq before.

The format data has been summarized in the file format2.tab:

Format file **format2.tab** from program 16.1

```
SELECT,         TYPE,      STYLEl l
RECHNUNG        div        vertikaleausrichtung:oben
BETREFF         span       rand-links:15mm
MITTE           span
UNTEN           div        position:absolute
                           oben:230mm
                           rand-links:10mm
                           border-width:6px
                           border-style:groove
                           border-color:darkgreen
ABSENDER        div        rand-oben:10mm
                           rand-links:15mm
                           rand-rechts:110mm
                           border-width:6px
                           border-style:groove
                           border-color:darkgreen
BILD            img        position:absolute
                           oben:10mm
                           rechts:15mm
                           border-width:6px
                           border-style:groove
                           border-color:violet
KUNDENADRESSE   div        rand-links:15mm
                           rand-oben:10mm
                           rand-unten:10mm
                           rand-rechts:110mm
                           border-width:6px
                           border-style:groove
                           border-color:cyan
L               span       border-width:6px
                           border-style:groove
                           border-color:darkgreen
M               span       border-width:6px
                           border-style:groove
                           border-color:darkgreen
R               span       border-width:6px
                           border-style:groove
                           border-color:darkgreen
TABELLEN        div        ausrichtung:rechts
                           rand-rechts:10mm
ZEILEN          tabelle    float:right
                           rand-oben:10mm
                           rand-rechts:10mm
                           schriftgroesse:gross
                           border-width:6px
                           border-style:groove
                           border-color:firebrick
ZAHLUNGSBETRAG  tabelle    float:right
                           clear:both
                           rand-oben:10mm
                           rand-rechts:10mm
                           schriftgroesse:gross
                           border-width:6px
                           border-style:groove
```

```
                            border-color:red
GESAMTPREIS                 ausrichtung:rechts
                            white-space:nowrap
STEUER                      ausrichtung:rechts
EINZELPREIS                 ausrichtung:rechts
MENGE                       ausrichtung:rechts
NAME             span       fett
BANKVERBINDUNG   div        position:absolute
                            oben:257mm
                            rand-links:15mm
                            border-width:6px
                            border-style:groove
                            border-color:darkgreen
```

For the calculation below, the image (the .png file) and the format file have been replaced with format.tab:

Backhaus Magdeburg
Inhaber: Manuel Schmidt
Breiter Weg 128
391 Magdeburg
Tel: 0391 22444
e-mail: backhausschmidt@web.de
Steuer-Nr. 103/2662/08839

Landgasthof Biederitz
Möser Str. 27
39175 Lostau

Re.Nr: 18#22
Rechn. vom 19.10.2022
Gerwisch, 21.10.2022

PRODUKT	ANZAHL	STEUER	PREIS	GESAMTPREIS
Baguette	6	7	2.20	14.12
Brötchen	77	7	0.34	28.01
Hanfbrot	6	7	3.50	22.47
Kürbisbrötchen	20	7	0.45	9.63
Milchbrötchen	26	7	0.43	11.96
Mischbrot 1 kg	7	7	1.90	14.23
Partystange	5	7	2.40	12.84

ZAHLUNGSBETRAG: 113.27€

Bankverbindung
Kontoinhaber: Manuel Schmidt
DE68 8106 3238 7777 7764 66
Volksbank Harzer Land

The same output is indicated with 2 background colors for demonstration purposes:

Backhaus Magdeburg
Inhaber: Manuel Schmidt
Breiter Weg 128
391 Magdeburg
Tel: 0391 22444
e-mail: backhausschmidt@web.de
Steuer-Nr. 103/2662/08839

Seniorendomizil Magdeburg
Vechelder Weg 10
39175 Biederitz

Re.Nr: 25#22
Rechn. vom 19.11.2022
Gerwisch, 21.11.2022

PRODUKT	ANZAHL	STEUER	PREIS	GESAMTPREIS
Baguette	6	7	2.20	14.12
Brötchen	77	7	0.34	28.01
Hanfbrot	6	7	3.50	22.47
Kürbisbrötchen	20	7	0.45	9.63
Milchbrötchen	26	7	0.43	11.96
Mischbrot 1 kg	7	7	1.90	14.23
Partystange	5	7	2.40	12.84

ZAHLUNGSBETRAG: 113.27€

Bankverbindung
Kontoinhaber: Manuel Schmidt
DE68 8106 3238 7777 7764 66
Volksbank Harzer Land

The file rechnungen22.hsq can be evaluated after each invoice creation for example with the following short program. Conditions could be inserted before the gib:

Program 19.2: Calculate total and monthly sales for each customer and product.

```
rechnungen22.hsq
MON:=DATUM text subtext 4!2
gib UMSATZ,(KUNDE,(UMSATZ,(MON,UMSATZ m)m)m),
    (PRODUKT,UMSATZ,(MON,UMSATZ m)m)m
UMSATZ:=ANZAHL*PREIS +% STEUER ! ++
rnd 2
```

Result (tab)

```
UMSATZ  ,(KUNDE       , (UMSATZ2, (MON, UMSATZ3 l) l) l),(PRODUKT               ,UMSATZ4,(MON,UMSATZ5 l) l)l
2066.10  AFF             254.21   11     195.42          Baguette                656.77  10     14.12
                                  12      58.79                                          11    157.72
         Backschwein     773.03   11     210.23                                          12    484.92
                                  12     562.80          Brötchen                200.82  10     28.01
         Landgasthof     113.27   10     113.27                                          11    107.32
         Senioren        925.58   11     634.88                                          12     65.48
                                  12     290.70          Brötchen belegt halb    231.01  11    120.05
                                                                                         12    110.96
                                                         Hanfbrot                827.65  10     22.47
                                                                                         11    554.26
                                                                                         12    250.92
                                                         Kürbisbrötchen           28.89  10      9.63
                                                                                         11     19.26
                                                         Milchbrötchen            35.89  10     11.96
                                                                                         11     23.93
                                                         Mischbrot 1 kg           46.56  10     14.23
                                                                                         11     32.32
                                                         Partystange              38.52  10     12.84
                                                                                         11     25.68
```

20 Appendix A: List of operations and keywords of o++o

Most of the known operations have an arity. The square root, for example, requires only one input value or argument - this is usually a number. In the o++o data model, this can also be a list of numbers. Then the square root is taken from each of the numbers. The list is then considered to be **one** input value, even though it may contain ten or even ten thousand numbers. That is, sqrt remains unary even in this case.

In the o++o syntax the sqrt symbol must follow the argument (postfix). This means that no additional parentheses are required. It is not allowed to write sqrt([2 4 7]) in o++o. But instead, you can use
[2 4 7] sqrt
or also
2 4 7 sqrt
.
In both cases you get the same result. You can even apply sqrt to any tabment.
Another example is the addition. The + operator is even better known than the root operation. It has arity 2, which means it requires two input values. The addition is binary. The application of the wrong number of arguments leads to a syntactical error and a corresponding error message.
3 +
as well as
3 4 +
lead to error messages.
In the term
3 + 4
3 is the first argument and 4 is the second input value. Again, a list or other tabment can be used as the first argument. The operation and the second argument are then applied to all elements of the list/table.
1 3 7 + 4
results in
5 7 11
Here and in many other operations the type of the result corresponds to the type of the first input table. So the above result is also a list of numbers. Binary operations are always written between the two input tables in o++o. You can also see it that they have to appear after the first input table like the unary operations. The same applies to three-digit operations in o++o. "!" is used as a separator between the second and third input values.

Hadmersleben subtext 4!5
for example, has the result:
mersl
The first input value is "Hadmersleben". The second input value (4) specifies the position of the initial letter of the partial word and the third input value (5) specifies the desired length.
5 if X>3 ! 6
also requires 3 input values (here: the 5, a truth value, and the 6). If we replace X by 10, the condition is fulfilled and the improved "if-then-else" operation returns 5. For X=1, however, the value 6 results.

In the following, the input and output data are illustrated once again using typical examples.

(first) Inputtabment	unary operation		Outputtabment
pi	sin	results in	0.
1 4 9	sqrt	results in	1. 2. 3.
1 4 9	++:	results in	4.66666666666

123456789	'3	results in	123'456'789

First input tab	binary operation	second input tab		Outputtabment
7	+	8	results in	15
7.	+	8	results in	15.
1 5 3	+	4	results in	5 9 7
1 5 3	+	1.2	results in	2.2 6.2 4.2
1 5 3	+	3 7 8	results in	4 12 11
1 5 3	+	3 7	not defined	
7 9	divrest	3	results in	2,1 3,0

First input tab	ternary operation	second input tab		third input tab		Outputtabment
"Georg Cantor"	subtext	1	!	5	results in	George
5	if	3=4 (no)	!	6	results in	6
1	...	2.9	!	0.5	results in	1. 1.5 2. 2.5
1	..x	2	!	4	results in	1 2 1 2

At this point it should be noted that in many cases the result of the previous line counts as the first input value of an operation:
marks.tab
++:
gives the average of all numbers that occur in the first line marks.tab. marks.tab is the input of ++:
The program
xx.tab
+ 2
adds 2 to each number in table xx.tab. xx.tab is the first input table and 2 is the second. In an analogous way extracts
names.tab
subtext 3!4
from each text value (TEXT or WORT or ONR) of names.tab a text of length 4 starting at the third position. Here the ternary subtext operation has the input tabs names.tab, 3 and 4.
In an assignment or condition several operations can be applied one after the other. If all operations are unary (one-line), then each corresponding one-line term has the form
tbt op_{11} op_{12} op_{13} ... op_{1n}
or more concretely:
1 2 3 sin abs sqrt ++text
If all operations are binary (two digits), the form is
tbt_1 op_{21} tbt_2 op_{22} tbt_3 op_{22} tbt_4 ... op_{2n} tbt_{n+1}
or more concrete
1 2 3 + 4 * 5 - 9
Terms with only 3 digit operations are certainly rare. Here is just a constructed example:

Operation symbol	Notation: Input→ Result type	Examples	Meaning

Magdeburg subtext 2!6 subtext 2!2
results in gd
If brackets are set, they must be calculated first:
abcdefghijk subtext 2!(2+3) results in bcdef
abcdefghijk subtext 2!2+3 on the other hand results in bc
(bc + 3 equals bc)
If you are not quite sure, you can put brackets as a precaution.
+ 3
is not a term, because the operation + has no first input value here. Therefore, an error message would appear. However, this would not be true if the above code were not on the first line. The value of the preceding line is then the first input value of + and 3 is then the second input value.
In the following, the designations below are used:
num = ZAHL or PZAHL or RATIO or BAR (|) or stroke list
nonum = TEXT or WORT or ONR
mixe = text and number occur in one column
tbt stands for any tabment type
For the types, we often specify only those that are also changed by the operation.

Operation symbol	Notation: Input→ Result type	Examples	Meaning
+	tbt1 + tbt2→ tbt1	1 1 3 + 2.1 results in 3.1 3.1 5.1 xy ab + de results in xyde abde	Addition of numbers and connecting text
*	tbt1 * num→ tbt1	2 3 5 * 2 results 4 6 10	Multiplication
-	tbt1 - num→ tbt1	3 - 2 results in 1 1'234 - 345 Results 889	Subtraction
:	tbt1 : num→ tbt1	3:4 results in 0.75	Division
++	tbt ++→ num	2 3 6 ++ results in 11	Total
**	tbt **→ num	1 3 5 ** results in 15	Product
--	tbt -- → num	20 5 4 -- results in 11	Multiple subtraction
::	tbt ::→ num	64 2 2 :: results in 16	Multiple Division
++:	tbt ++:→ PZAHL	1 2 3 2 ++: results 2.0	Average
++1	tbt ++1→ NUMBER	3 4 7 9 ++1 results in 4	Quantity
++text	tbt ++text → text	[ab cde fg] ++text results in abcdefg	Connect to text
++textsep	TEXTl ++textsep "sep"→ TEXT	ab cde fg ++textsep ";" yields "ab;cde;fg"	combine to text with inclusion of a separator

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
,	tbt1,tbt2→ tbt	1 2,3 results in COUNTl,COUNT 1 3 2	Pairing
;	tbt1;tbt2→ tbt	2.3 * 2 equals 4.6 2;3 *2 gives 2.6	also a pair formation. but ; separates sharper than ,
,,	tbt1,,tbt2 → tbt	1 3 ,, xx yy results in 1 xx 3 yy	Multiple pair formation
=	tbt1,tbt2→ BOOL	1 = 2 results in si	Equality
<=	tbt1 <= tbt2→ BOOL	2 <= 2 results si	Less than or equal to
>=	tbt1 >= tbt2→ BOOL	2 >= 4 results no	Greater than or equal to
+coll	coll1 +coll coll2→ coll1	{{1 2}} +coll {1} results in {{1 1 2 }}	"set-theoretic" union
-coll	coll1 -coll coll2→ coll1	[2 4 3 2] -coll [2] results in 4 3 2	Set difference
*coll	coll1 *coll coll2→ coll1	{1 2 3} *coll {4 5} results in ZAHL,ZAHL m 1 4 1 5 2 4 2 5 3 4 3 5	Cartesian product
:coll	coll1:coll	{1 2 3}:coll [2 3 4] results in	Intersection

<table>
<tr><th>Operation symbol</th><th>Notation: Input→ Result type</th><th>Examples</th><th>Meaning</th></tr>
<tr><td></td><td>coll2→ coll1</td><td>{2 3}</td><td></td></tr>
<tr><td>*l</td><td>tbt *l NUMBER → tbt l</td><td>car *l 3
results in
car car car
or
xx.tab *l 3</td><td>Multiply element to list</td></tr>
<tr><td>*mat</td><td>coll1 *mat coll2→ coll</td><td>(1,2) *mat [2 3]
results in
8</td><td>Matrix multiplication</td></tr>
<tr><td>-1mat</td><td>coll -1mat→ coll</td><td><TAB!
X1,X2,X3 l
1 0 2
0 2 0
0 0 8
!TAB>
-1mat
results in
X1, X2, X3 l
1. -0. -0.25
-0. 0.5 -0.
0. -0. 0.125</td><td>inverse matrix</td></tr>
<tr><td>&</td><td>BOOL & BOOL -> BOOL</td><td>si & no
results in
no</td><td>Conjunction (logical and)</td></tr>
<tr><td>|</td><td>BOOL | BOOL -> BOOL</td><td>1=1 | 1=2
results in
si</td><td>Disjunction (logical or)</td></tr>
<tr><td>&&</td><td>coll1 &&→ BOOL</td><td>si,66,si &&
results si</td><td>for all</td></tr>
<tr><td>||</td><td>coll1 ||→ BOOL</td><td>1=2,no ||
results no</td><td>it exists</td></tr>
<tr><td>|l</td><td>ZAHL |l -> BARl</td><td>5 |l
results in
| | | | |</td><td>Transfer numbers into tally sheets (for kindergarten)</td></tr>
<tr><td>..</td><td>number1 ..</td><td>1 .. 4
results in</td><td>from ... to</td></tr>
</table>

Operati on symbol	Notatio n: Input→ Result type	Examples	Meaning
	number2 -> numberl	1 2 3 4	generate
...	number1 ... number2 ! number3 → ZAHLl	0 ... 6!2 results in 0 2 4 6	from ... to ! step
..x	number1 ... number2 ! ZAHL -> numberl →	1 ..x 6!3 results in 5 3 2	Random numbers from ..x to ! number
'3	tbt '3→ tbt	1234567890 '3 results in 1'234'567'890	format in blocks of 3
'4	tbt '4→ tbt	12345.67898 '4 results in 1'2345.6789'8	format in blocks of 4
^ hoch	tbt ^ num→ tbt	4 ^ 1/2 results in 2. 10 *l 4 to the power of (0 ..3) results in 1 10 100 1000	to the power of
abs	tbt abs→ tbt	-3 7 abs results in (tabh) 3 7	absolute amount
aggseg	tbt aggseg op ! NAME	[(1,2) (3,4)] aggseg ++ | ZAHL results in: ZAHL, ZAHL l 1 2 3 4 4 6	vertical op- aggrgation for all NAME- collection s
aggsegs	tbt aggsegs op -> tbt	[(1,2) (3,4)] aggsegs ++ results in ZAHL, ZAHL l	In all collection s will be verical

Operation symbol	Notation: Input→ Result type	Examples	Meaning
		1 2 3 4 4 6	aggregated
arctan	tbt arctan -> tbt	1 arctan gives 0.785398163397 (= pi:4)	arcus tangent
at		GROSS:=NET +% 19 at NET	place new column to the right of the specified
aus	tbt1 from tbt2 → tbt2	aus rivers.tabh	New start of a program
avec	tbt1 avec bed → tbt1	rivers.tabh avec LENGTH >800	Selection (with)
comp	tbt name→ tbt	<TAB! NAME,FIRSTNAME,PLACE Mill Paul Halle !TAB> comp PLACE results in PLACE Halle (see also nth)	Component
cos	PZAHL cos→ PZAHL	pi cos results in -1.	Cosine
cut	TEXT cut ZAHL -> List	qwertzuiop cut 3 results in qwe rtz uio p	cut a text in a list of text of equal length
cutsplit	TEXT cutsplit ZAHL -> List	"qw ert zuio" cutsplit 3 results in qw ert zui o	cut a text in pieces of equal length, where the cut is

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
			earlier made when a blank exists
det	coll det→ coll	<TAB! X1,X2,X3 l 1 0 2 0 2 0 0 0 8 !TAB> det results in 16.	Determinant
div	NUMBER div NUMBER → NUMBER	11 div 5 results in 2	integer division
divrest	NUMBER divrest NUMBER → Pair	11 divrest 5 results in 2,1 (not 2.1)	integer division with remainder
euler	euler	euler ^ 3 ln results in 3.	Euler's constant
falls	term falls cond → tbt1	1 if 4=4 results in 1? 1 falls 4=3 results in empty optional value	if with 2 input values, in connection with gib certain joins can be expressed in this way
for		X:= 100 for X pred *1.03 at Y	precedes the second term of a recursive assignment
foronr		X:=firstonr 100 foronr X pred *1.03 at Y	for for onr recursion

Operation symbol	Notation: Input→ Result type	Examples	Meaning
gib	tbt1 gib schema → tbt2	aus students.tab gib FAC,(LOC,NAMEm)m	Restructuring of a tabment regarding given schemas and aggregations
gibl	tbt1 gibl schema -> tbt2	gibl X,Ym- m results in tabment with the scheme X,Yl l with same content as gib	Is just a shorthand notation for 2 gib statements
giball	tbt1 giball scheme2 → tbt2	giball X | Y l List of all X and Y subtab segments (any depth); corresponds to ...//X|Y of XPath	Extraction of all corresponding values
gibtop	tbt1 gibtop scheme2 → tbt2	gibtop Xl corresponds to: t/X: List of all X-Subtabmente of t, from the highest level of t.	Extraction of the top values
hori	tbt hori name→ tbt'	<TAB! SUBJECT,NOTE m Ger 1 Phy 2 Ma 1 !TAB> hori SUBJECT results in GER, MA, PHY 1 1 2	Arrange data horizontally
if	tbt if condition !tbt2	1 if 1=2 ! 3 results in 3	ternary operation that replaces "if_then-else" and "case".
igib	tbt1 igib scheme2 → tbt2	students.tab,faks.tab igib FAK,DEAN,NAMEm m	Join with restructuring
in	tbt1 in tbt2→	"1 2 1" in "1 2" results in si	Every word of the

Operation symbol	Notation: Input→ Result type	Examples	Meaning
	BOOL	"1 2 3" in "1 1 2" results no	left side is word of the right side
inmath	tbt1 inmath tbt2→ BOOL	[1 3] inmath [1 4 3] gives si [3 1] inmath [1 4 3] gives no 2 inmath {6 7 2} gives si	mathematical inclusion; the left hand side determines, whether we have a list, set or bag inclusion
keys	tbt1 keys tbt2→ tbt1	Xl:= 1 ..40 Y:=X*X gib X,Y m keys [7 34] results in tab format: X, Y m 7 49 34 1156 or keys [yy,[y2] zz]	Efficient selection in sets or lists
keyslike	tbt1 keyslike tbt2→ tbt1	<TAB! NAME, PLACE m Clara Oehna Claudia Dallgow Sophia Dallgow !TAB> keyslike ["*ia"] results NAME, PLACE m Claudia Dallgow Sophia Dallgow	efficient selection in sets or lists with partial matching
leftat		GROSS:=NET +% 19 leftat NET	place new column to the left of the specified one
like	term like "term?*"	Hadmersleben like "?admers*" results si '?': represents a character '*': represents 0 or more characters	similar to

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
	→ BOOL		
linreg	tbt linreg → num,num	<TAB! PRICE,SOLD l 20 0 16 3 15 7 16 4 13 6 10 10 !TAB> linreg results in 19.73214,-0.98214	linear regression
list	tbt list -> tbt	pi,euler,3.14 list results in (tab format) 3.14159265359 2.71828182846 3.14	A tuple is converted into a list
lists	tbt lists zahl → tbt l	Xl:= 1 2 lists 2 results (tabh format) Xl l 1 1 1 2 2 1 2 2	Generate a list of lists of specified length
ln	tbt ln→ PZAHL	euler ln results in 1.	natural logarithm
log	tbt1 log tbt2→ PZAHL	100 log 10 results in 2.	general logarithm
lower	tbt lower→ tbt	AsdRRGee34 lower results in asdrrgee34	turn into lowercase letters
max	tbt max→ num	12.21,2,Hello max results in 12.21	Maximum of all numbers
median	tbt median → num	1 2 4.9 median results in 3.0	Median

Operation symbol	Notation: Input→ Result type	Examples	Meaning
min	tbt min→ num	12.21,2,Hello min results in 2	Minimum of all numbers
minus	tbt minus -> tbt	1 -2 4 minus results in -1 2 -4	negate any number
natsel	tbt natsel -> tbt	students.tab,exams.tab avec NAME=Ernst natsel exams then also contains only exams from Ernst	natural selection (regarding common column names)
no	no→ BOOL	no | si results si	Truth value false corresponds to the answer no (Spanish no)
not	BOOL not→ BOOL	si not results no	Negation
nth	tbt nth NUMBER → tbt'	1 3 5 nth 2 results in 3	nth component
nthpred	Name nthpred NUMBER → term	Xl:= 1 2 3 4 Y:= X nthpred 2 results in X,Y? l 1 2 3 1 4 2	n-th predecessor
nthsucc	tbt nthsucc NUMBER → tbt'	Xl:=2 4 7 4 3 4 4 avec X nthsucc 2=4 results in (tabh) 4 4 3	n-th successor
nthzahl	tbt nth number NUMBER → tbt'	"2023.03.26" nthzahl 3 results in 26	nth number in a text
onr	tbt	1 3 5.2 "4.5.5" onr	Conversion

Operation symbol	Notation: Input→ Result type	Examples	Meaning
	onr→ tbt	results (tabh): 1 3 5.2 4.5.5	to o++o number

onrs	tbt
	onrs
	name !
	element
	→ tbt'
		<TAB!
		X,Ym m
		k y
		  z
		y w
		!TAB>
		onrs OTTO!k
		results
		X, (OTTO, Y m) l
		k 1 z
		    2 y
		    2.1 w
			generates o++o numbers in a table; this is an important component of BOM explosion.

permutations	list permutations -> listl	2 4 9 permutations results in (tabh) ZAHLl l 2 4 9 2 9 4 4 2 9 4 9 2 9 2 4 9 4 2	"permutations" is an abbreviation for the program: Xl:= 2 4 9 lists 3 avec Xm= {2 4 9}
pi	pi→ PZAHL	CIRCULAR AREA:=R*R*pi	Circle number
poly	num poly list→ num	3 poly [1 2 3] results in 18	Polynomial
pos	Name pos→ NUMBER	avec X pos < 10	Position
pos-	Name pos-→ NUMBER	avec X pos- > 5	Position from behind
pred	Name pred→ term	X:= 100 for X pred *1.03	Predecessor
preds	preds → tbt		Abbreviation of X pred, Y pred, …
primo	tbt	1 3;4;7 8 9	Select

Operation symbol	Notation: Input→ Result type	Examples	Meaning
	primo -> tbt	primo results in (tab) 1 4 7	first element of each collection
pzahl	tbt pnumber → PZAHL	1/5 6 9.7 pzahl results in (tabh) 0.2 6. 9.7	Conversion to a PZAHL
pzahl1de	tbt pzahl1de→ PZAHL	"Today I get 356.88 euros and not 66.8 ." pzahl1de results in 356.88	First PZAHL of a German text

rat	ZAHL rat ZAHL→ RATIO	<TAB!
		X,Yl l
		1 2
		  3
		!TAB>
		Z:= X rat Y
		results in
		X,(Y, Z l) l
		1 2 1/2
		   3 1/3
			Conversion of two numbers into one RATIO number

ratio	num ratio→ RATIO	1/5 6 9.7 ratio results in (tabh) 1/5 6/1 97/10	Conversion to rational number
rename	tbt rename Name1 ! name2→ tbt'	rename X!Y	Renaming column names
rest	NUMBER rest NUMBER → NUMBER	13 rest 5 results in 3	Remainder for integer division
rnd	PZAHL rnd ZAHL→ PZAHL	17,678 3.45 zz 8 rnd 1 results in 17.7 3.5 zz 8	round
route	tbt route→ tbt	<TAB! X,Y m 0 0	Generate straight line

<table>
<tr><th>Operation symbol</th><th>Notation: Input→ Result type</th><th>Examples</th><th>Meaning</th></tr>
<tr><td></td><td></td><td>1 1
0 1
!TAB>
route
generates 2 lines from (0,0) to (1,1) and from (1,1) to (0,1)</td><td>sequence from point sequence</td></tr>
<tr><td>sans</td><td>tbt
sans
cond
→ tbt</td><td>sans LOC=Magdeburg
sans Magdeburg
sans: without the specified struples</td><td>Selection (without)</td></tr>
<tr><td>satzl</td><td>TEXT
satzl
TEXTl</td><td>"It's great. Great. Tomorrow we celebrate."
satzl
results (tabh-format):
SATZl
It's great.
Great.
Tomorrow we celebrate.</td><td>List of sentences</td></tr>
<tr><td>seg</td><td>NAME
seg→
term</td><td><TAB!
X,Y,Z,U,Vl l
1 2 3 4 5
6
6 7 8 9 10
11
!TAB>
SUM:= Z seg ++
results in (tabh)
X ,Y ,Z ,SUM ,U ,Vl l
1 2 3 7 4 5 6
6 7 8 17 9 10 11</td><td>segment from position NAME</td></tr>
<tr><td>sepl</td><td>constant list, useful to understand exactly the operation cil (split into words)</td><td>"." ";" "," "|" "!" "?" "(" ")" "@" "#" "\n" "-" " "</td><td>all separators used in the cil operation</td></tr>
<tr><td>si</td><td>si→
BOOL</td><td>si & no
results in
no</td><td>Truth value true (answer yes (=si))</td></tr>
<tr><td>sin</td><td>PZAHL
sin→</td><td>3.14159 sin
results in</td><td>Sine function</td></tr>
</table>

<table>
<tr><th>Operation symbol</th><th>Notation: Input→ Result type</th><th>Examples</th><th>Meaning</th></tr>
<tr><td></td><td>PZAHL</td><td>2.65358979335e-06</td><td></td></tr>
<tr><td>split</td><td>TEXT split "sep"→ TEXTl</td><td>Xl:= "Brati,Novi Sad, Belgrade" split ","
result (ment):
<TABM>
  <X>Brati</X>
  <X>Novi Sad</X>
  <X>Belgrade</X>
</TABM></td><td>Decompose text</td></tr>
<tr><td>splitfull</td><td>tbt splitfull→ WORTl</td><td>"We live in Halle, a city in Saxony-Anhalt". splitfull
results (ment)
<TABM>
We
live
in
Hall
a
City
in
Saxony
Anhalt
</TABM></td><td>List of all words, where the following separators are given:
" " . , ;
- | / ! ?
( ) @ #
"\n" (end of line)</td></tr>
<tr><td>splitfullm</td><td>tbt splitfullm→ WORTm</td><td>"We live in Halle, a city in Saxony-Anhalt". splitfullm
results in (tabh)
WORTm
anhalt hall saxony city we one in live</td><td>Set of all words, where the following separators are given:
" " . , ;
- | / ! ?
( ) @ #
"\n" (end of line)</td></tr>
<tr><td>splitsep</td><td>TEXT splitsep "sep"→ TEXTl</td><td>Xl:= "Brati,Novi Sad, Belgrade" splitsep ","
result (ment):
<TABM>
  <X>Brati</X>
  <X>,</X>
  <X>Novi Sad</X>
  <X>,</X>
  <X> Belgrade</X>
</TABM></td><td>Decompose text, not deleting the separator</td></tr>
<tr><td>sqrt</td><td>num sqrt→ PZAHL</td><td>4 sqrt
results in
2.</td><td>Square root</td></tr>
</table>

Operation symbol	Notation: Input→ Result type	Examples	Meaning
mad streu	tbt scatter → PZAHL	[1 2 5 3 5 1] mad results in 1.5	Scattering
subtext	text subtext NUMBER ! NUMBER → TEXT	aBCdE subtext 2 ! 3 results in BCd	Subtext (substring)
subtext 2	text subtext 2 text ! text→ TEXT	aBCdEfgh subtext2 "B"!fg results in CdE	Partial text of the first text that lies between the other two given texts.
succ	Name succ→ term	NOTEl:= 3 1 2 1 avec NOTE >NOTE succ results in NOTEl 3 2	Successor
tag	tbt tag NAME!scheme → tbt'	LOCATION:=Magdeburg STREET:=Beims tag ADDRESS!LOCATION,STREET results (metadata) TABMENT ! ADDRESS ADDRESS ! LOCATION,STREET LOCATION ! WORT STREET ! WORT	enclose data of a schema with a tag
tag0	tbt tag0 name→ tbt'	11 13 tag0 XX results (ment) <XX> 11 13 </XX>	Put a tag around the entire tabment
tan	num tan→ PZAHL	3.14 tan results in -0.00159265493641	Tangent function
text	mixe text→ TEXT	3.14 ttt 8 text results in TEXTl	Transform any elementary type to

Operation symbol	Notation: Input→ Result type	Examples	Meaning
		3.14 ttt 8	TEXT.
textend	tbt textend NUMBER → TEXT	asdfgh text end 4 results fgh abcde textend -2 Results de	subtext counted from back
textindex	text mixe text→ NUMBER	"Today is Tuesday." text index Di results in NUMBER 11	Position
time	time→ PZAHL	time could result: 1.557021	system time (only the difference between two such times is significant for efficiency considerations)
trim	text trim→ text	" Hi o++o " trim results (ment) <TABM>Hi o++o</TABM>	remove spaces at the back and front
tup	NAME tup→ term	<TAB! X,Y,Z,U,Vl l 1 2 3 4 5 6 6 7 8 9 10 11 !TAB> SUM:= Z tup ++ Results in (tabh) X ,Y ,Z ,SUM ,U ,Vl l 1 2 3 18 4 5 6 6 7 8 38 9 10 11	a whole struple from position NAME
ultimo	tbt ultimo -> tbt	1 2 4, 5 ultimo results in (tab) ZAHLl, ZAHL 4 5	from each collection preserve only the last element
untag0	tbt	X:=1	remove the

Operation symbol	Notation: Input→ Result type	Examples	Meaning
	untag0 → tbt'	untag0 results in NUMBER 1	outermost day
upper	text upper→ text	1.2,aW upper results (tab-format) PZAHL, WORD 1.2 AW	convert to uppercase
variance	tbt variance→ PZAHL	[1 2 4 6] variance results in 4.91666666667	Variance
verti	tbt verti coll:=tup→ tbt'	verti MON,XX l:=JAN ..DEC verti SUBJECT,MARKl l:= PHYl ..MATHl	Arrange data vertically
vlists	tbt vlists ZAHL→ tbt l	variable-length lists; the operation generates the same as "lists" except that all shorter lists also appear in the result.	Variable length lists
weg	tbt weg names → tbt'	<TABH! X,Ym m 1 2 3 4 5 !TABH> weg Y results (tab-format) Xm 1 4	Omitting columns
wort	tbt wort→ wort	"I'm good.So are you." word results in WORT I_am_good.You_too.	Convert to words
zahl	num zahl→ ZAHL	"12" zahl results in 12 3.14 zahl results in 3	Conversion
zahltri	text	DAY,MON,YEAR:=	The first

Operation symbol	Notation: Input→ Result type	Examples	Meaning
p	zahltrip -> Triples of numbers	26.03.1963 zahltrip results in DAY,MON,YEAR 26 3 1963	3 numbers of a text
zil	tbt zil→ tbt'	123,Today zil results in (tabh) 1 2 3, H e u t e	List of all characters (letters+digits+ special characters) (zi Chinese: character)
zahlratio	RATIO zahlratio -> ZAHL,RATIO	33/7 zahlratio results in 4 5/7	Convert to integer part and real fraction
zahl1de	text number1en→ NUMBER	"Today I get 66,356.11 euros". zahl1de results in 66356	extract first number from a German text

21 Appendix B: Grammar

```
%start main
%token ABS          abs
%token ADDAGGS      addaggs
%token ALL          &&
%token ALLSEGS      allsegs
%token AND          &
%token APO3         '3
%token APO4         '4
%token ARCTAN       arctan
%token AT           at
%token ATOM         atom
%token AUS          aus
%token AUSRUFE      !
%token AVEC         avec
%token AVG          ++:
%token BAR          |
%token BARL         |l
%token BE           hoch ^
%token BEGIN        begin
%token BOOL
%token CART         *coll
%token COLL         l m b l- m- b- a ?
%token COMMA        ,
%token COMP         comp
%token CONTENT1
%token CONTENT2
%token COS          cos
%token COUNT        ++1
%token COUNTSTROKE ++|
%token CREATE       create
%token CSV          csv
%token CSVTABLE
%token CUT          cut
%token CUTSPLIT     cutsplit
%token DCOMMA       ,,
%token DDDOT        ...
%token DDOT         ..
%token DDOTX        ..x
%token DEFOP        defop
%token DELETE       delete
%token DET          det
%token DIV          :
%token DIVBIGINT    div
%token DIVDIV       ::
%token DIVREST      divrest
%token DOLLAR       $
%token DOLLAR2      $$
%token DSEMI        ;;
%token END          end
%token EOF
%token EOL          ; "\n"
%token EQ           =
```

```
%token EQ2         ==
%token EQUI        <->
%token EULER       euler
%token EX          ||
%token EXCEPT      -coll
%token FALLS       if
%token FLOAT       float
%token FUNNAME     myop. ..
%token GE           >=
%token GE2          >=>=
%token GIB          gib
%token GIBALL       giball
%token GIBL         gibl
%token GIBTOP       gibtop
%token GT           >
%token GT2          >>
%token HORI         hori
%token HSQ          hsq
%token HSQH         hsqh
%token HSQTABLE
%token HSQTABLEH
%token IF           if
%token IGIB         igib
%token IMPLI        ->
%token IMPLIR       ::=
%token IN           in
%token INCLUDE      include
%token INMATH       inmath
%token INSERT       insert
%token INSIDE       inside
%token INTEGER
%token INTERSECT    :coll
%token INVERSMAT    -1mat
%token IS           :=
%token JPG          jpg
%token JSON         json
%token JSONTABLE
%token KEYS         keys
%token KEYSLIKE     keyslike
%token LBRACK       [
%token LCURL        {
%token LCURL2       {{
%token LE           <=
%token LE2          <=<=
%token LEFTAT       leftat
%token LETTERL      zil
%token LIKE         like
%token LINREG       linreg
%token LISTS        lists
%token LN           ln
%token LOAD         load
%token LOG          log
%token LOWER        lower
%token LPAREN       (
%token LT           <
```

```
%token LT2          <<
%token MAD          mad streu
%token MAL          *l
%token MANT         mant
%token MAX          max
%token MEDIAN       median
%token MENT         ment
%token MENTTABLE
%token MIN          min
%token MINUS        -
%token MINUSMINUS   --
%token MINUSOP1     minus
%token MINUSPLUS    -+
%token MINUSPROZENT -%
%token MIXE
%token MOD          rest
%token MULT         *
%token MULTKOMPLEX  *i
%token MULTMAT      *mat
%token NAME
%token NAMEC         ..m  ..l ..b
%token NAMECMINUS    ..m-   ..l-  ..b-
%token NATJOIN       natjoin
%token NATSEL        natsel
%token NE            !=
%token NE2           !=!=
%token FOR           for
%token FORONR        foronr
%token NINTEGER
%token NO            no
%token NORM10E       norm3e
%token NORM10M       norm3m
%token NORMT         normt
%token NOT           not
%token NTH           nth
%token NTHZAHL       nthzahl
%token NUMMER        num
%token NUR           nur
%token ONEIN         1in
%token ONEINMATH     1inmath
%token ONR           onr
%token ONR2          onr2
%token ONRS          onrs
%token OR            |
%token PATHNAME
%token PERMUTATIONS  permutations
%token PI            pi
%token PLUS          +
%token PLUSPLUSTEXT  ++text
%token PLUSPLUSTEXT2 ++textsep
%token PLUSPROZENT   +%
%token POLY          poly
%token POS           pos
%token POS2          pos-
%token PRED          pred
```

```
%token PREDTUP      predtup
%token PREDS        preds
%token PRED_N       nthpred
%token PRIMO        primo
%token PROD         **
%token PROZENT      %
%token PZAHL        pzahl
%token PZAHL1DE     pzahl1de
%token RAM          ram
%token RAT          rat
%token RATIO        ratio
%token RATIOTYPE    ratio
%token RBRACK       ]
%token RCURL        }
%token RCURL2       }}
%token RENAME       rename
%token REONR        reonr
%token RGB          rgb
%token ROTATE       rotate
%token ROUND        rnd
%token ROUTE        route
%token RPAREN       )
%token RPOS         pos-
%token RPOS2        rpos2
%token SANS         sans
%token SATZL        satzl
%token SAVE         save
%token SEG          seg
%token SEMI         ;
%token SEPL         sepl
%token SI           si
%token SIN          sin
%token SPLIT        split
%token SPLITSEP     splitsep
%token SQRT         sqrt
%token STANDARD     standard
%token STRICH       ~
%token STRING
%token STRING2
%token STRING3
%token SUBTEXT      subtext
%token SUBTEXT2     subtext2
%token SUCC         succ
%token SUCCTUP      succtup
%token SUCC_N       nthsucc
%token SUM          ++
%token TAB          tab
%token TABH         tabh
%token TABLE
%token TABLEH
%token TAG          tag
%token TAG0         tag0
%token TAGALL       tagall
%token TAN          tan
%token TEXT         text
```

```
%token TEXTEND       textend
%token TEXTINDEX     textindex
%token TIME          time
%token TOANY         2any
%token TRANSPOSE     transpose
%token TREE          tree
%token TRIM          trim
%token TUP           tup
%token TXT           txt
%token TXTTABLE
%token ULTIMO        ultimo
%token UNION         +coll
%token UNTAG         untag
%token UNTAG0        untag0
%token UNTAGALL      untagall
%token UPDATE        update
%token UPPER         upper
%token VARIANCE      variance
%token VERTI         verti
%token VLISTS        vlists
%token WAS           =:
%token WEG           weg
%token WEGE          wege
%token WEGECYC       wegecyc
%token WHILE         while
%token WIKI          wiki
%token WORT          wort
%token WORTL         cil
%token WORTM         cim
%token XML           xml
%token XMLTABLE
%token ZAHL          zahl
%token ZAHL1DE       zahl1de
%token ZAHLRATIO     zahlratio
%token ZAHLTRIP      zahltrip

%left SEMI
%left AND EQUI IMPLI OR
%left EQ GE GT IN INMATH LE LIKE LT NE ONEIN ONEINMATH
%left ABS ADDAGGS ALL APO3 APO4 ARCTAN AVG BARL BE CART COLL COMMA COMP COS
COUNT COUNTSTROKE CREATE DCOMMA DDDOT DDOT DDOTX
  DELETE DET DIV DIVBIGINT DIVDIV DIVREST EQ2 EX EXCEPT FALLS
  FILLPOLYGON FUNNAME GE2 GIB GIBALL GIBTOP GT2 HORI INSERT INSIDE
INTERSECT INVERSMAT KEYS KEYSLIKE LE2
  LETTERL LINREG LISTS LN LOAD LOG LOWER LT2 MAD MAL MANT MAX MEDIAN MIN
MINUS MINUSMINUS MINUSOP1
  MINUSPLUS MINUSPROZENT MOD MULT MULTKOMPLEX MULTMAT NATJOIN NATSEL NE2
NORM10E NORM10M NORMT NOT NTH NTHZAHL
  NUMMER ONR PERMUTATIONS PLUS PLUSPLUSTEXT PLUSPLUSTEXT2 PLUSPROZENT POLY
PRED_N PRIMO PROD PROZENT PZAHL PZAHL1DE RAT RATIO
  REONR RGB ROTATE ROUND ROUTE SATZL SAVE SIN SPLIT SPLITSEP SQRT STANDARD
SUBTEXT SUBTEXT2 SUCC_N SUM TAG TAG0 TAGALL TAN
  TEXT
  TEXTEND TEXTINDEX TOANY TRANSPOSE TREE TRIM ULTIMO UNION UNTAG UNTAG0
UNTAGALL UNTAGTOPEXCEPTFORMAT UPDATE UPPER VARIANCE
```

```
  VERTI VLISTS WEGE WEGECYC WORT WORTL WORTM ZAHL ZAHL1DE ZAHLRATIO
ZAHLTRIP CUT CUTSPLIT
%type <unit> Agg
%type <unit> Bars
%type <unit> Command
%type <unit> Command2
%type <unit> CommandList
%type <unit> CommandListCore
%type <unit> Expr
%type <unit> Expr_list
%type <unit> FpathName
%type <unit> FpathNames
%type <unit> Istroke
%type <unit> Name2
%type <unit> Names
%type <unit> Oper1
%type <unit> Oper2
%type <unit> Oper2_all
%type <unit> Oper2_lt
%type <unit> Oper2_mal
%type <unit> Oper2_name
%type <unit> Oper2_names
%type <unit> Oper2_or
%type <unit> Oper2_plus
%type <unit> Oper2_ram
%type <unit> Oper2_scheme
%type <unit> Oper2_scheme_standalone
%type <unit> Oper2_textsep
%type <unit> Oper2_union
%type <unit> Oper3
%type <unit> Oper3_onrs
%type <unit> Oper3_scheme
%type <unit> Oper3_zufall
%type <unit> Operfolge
%type <unit> PathName
%type <unit> PathNames
%type <unit> Ram_expr
%type <unit> Scheme
%type <unit> SchemeIs
%type <unit> Sel3
%type <unit> Simpleagg
%type <unit> Strich
%type <unit> Stroke
%type <unit> Tableexpr
%type <unit> Tableexpr_c
%type <unit> Tableexpr_file
%type <unit> Trenner
%type <unit> Value
%type <unit> Valuelist
%type <unit> Where2
%type <unit> main
%%

main:
  CommandList EOF
```

```
| EOF

CommandList:
  CommandListCore
| Trenner CommandList

CommandListCore:
  Command
| CommandListCore Trenner Command2
| CommandListCore Trenner

Command:
  Command2
| Expr

Trenner:
  EOL
| DSEMI

Command2:
  AUS Expr
| AUS SchemeIs Expr
| DEFOP DOLLAR NAME FUNNAME EQ Command
| INCLUDE Expr
| WAS DOLLAR NAME
| WAS DOLLAR2 NAME
| DOLLAR NAME IS Expr
| DOLLAR2 NAME IS Expr
| SchemeIs Expr Where2
| Operfolge
| Oper2_all Expr
| Oper2_name NAME
| Oper2_names PathNames
| Oper3_scheme Scheme IS Scheme DDOT Scheme
| TAG NAME AUSRUFE Scheme
| Oper3_onrs NAME AUSRUFE Expr
| Oper2_ram RAM
| Oper2_scheme_standalone Scheme
| Oper3 Expr AUSRUFE Expr
| ADDAGGS NAME AUSRUFE Oper1
| SAVE Tableexpr_file
| Stroke
| Istroke
| SchemeIs Expr FOR Expr Where2
| SchemeIs Expr FORONR Expr Where2
| SchemeIs Expr WHILE Expr AUSRUFE Expr Where2
| Sel3 Expr
| Sel3 PathNames AUSRUFE Expr
| PathName IMPLIR Expr
| RENAME PathName AUSRUFE NAME
| WEG PathNames
| NUR PathNames
| Agg Expr

Where2:
```

```
| AT FpathNames
| LEFTAT FpathNames

Sel3:
  AVEC
| SANS

Stroke:
  GIB Scheme
| Stroke NAME EQ Scheme
| Stroke ATOM DIV Scheme
| Stroke NAME IS Expr AUSRUFE Agg

Strokel:
  GIBL Scheme
| Strokel NAME EQ Scheme
| Strokel ATOM DIV Scheme
| Strokel NAME IS Expr AUSRUFE Agg

Istroke:
  IGIB Scheme
| Istroke NAME IS Expr AUSRUFE Agg
| Istroke NAME AUSRUFE Simpleagg

Scheme:
  Scheme COLL
| LPAREN Scheme RPAREN
| Scheme COMMA Scheme %prec PLUS
| Scheme OR Scheme %prec PLUS
| Name2
| MIXE
| NAMEC
| NAMECMINUS

Agg:
  Simpleagg
| AVG
| COUNT
| VARIANCE
| MAD
| MEDIAN
| LINREG
| COUNTSTROKE

Simpleagg:
  SUM
| MAX
| MIN
| PROD
| EX
| ALL

Names:
  NAME COMMA Names
| NAME
```

```
SchemeIs:
  Scheme IS

PathName:
  PATHNAME
| NAME

PathNames:
  PathName PathNames
| PathName

FpathName:
  PATHNAME
| NAME
| NAMEC

FpathNames:
  FpathName FpathNames
| FpathName

Name2:
  NAME
| ZAHL
| PZAHL
| RATIO
| TEXT
| WORT
| BOOL
| BAR
| ONR

Ram_expr:
  WIKI

Expr_list:
  Expr
| Expr_list Expr

Expr:
  NAME
| PATHNAME
| NAME Strich
| NAME PREDTUP
| NAME SUCCTUP
| NAME SUCC_N Expr
| NAME PRED_N Expr
| NAME PRED
| NAME SUCC
| Ram_expr
| Expr Oper2_ram RAM
| Expr SAVE NAME
| Expr Oper2_name NAME
| Expr Oper2_names Names
| Expr Oper3_scheme Scheme IS Scheme DDOT Scheme
```

```
| Expr TAG NAME AUSRUFE LPAREN Scheme RPAREN
| DOLLAR NAME
| DOLLAR2 NAME
| TIME
| LPAREN Expr RPAREN
| LBRACK Expr_list RBRACK
| LBRACK Bars RBRACK
| LCURL Expr_list RCURL
| LCURL2 Expr_list RCURL2
| LBRACK RBRACK
| LCURL RCURL
| LCURL2 RCURL2
| LBRACK Valuelist RBRACK
| LCURL Valuelist RCURL
| LCURL2 Valuelist RCURL2
| LBRACK Names AUSRUFE Expr DDOTWHILE Expr RBRACK
| LBRACK Names AUSRUFE Expr DDDOTWHILE Expr AUSRUFE Expr RBRACK
| Expr IF Expr
| Expr Oper3 Expr AUSRUFE Expr %prec PLUS
| Expr ADDAGGS NAME AUSRUFE Oper1 %prec PLUS
| OR MAL Expr
| Expr Oper2_or Expr %prec AND
| Expr Oper2_lt Expr %prec LT
| Expr Oper2 Expr %prec PLUS
| Expr Oper1
| Expr FUNNAME
| Expr Oper2_scheme Scheme
| NAME POS
| PATHNAME POS
| NAME RPOS
| PATHNAME RPOS
| NAME POS2
| PATHNAME POS2
| NAME RPOS2
| PATHNAME RPOS2
| NAME TUP
| PATHNAME TUP
| NAME SEG
| PATHNAME SEG
| NAME ALLSEGS
| PATHNAME ALLSEGS
| NAMEC
| NAMECMINUS
| BEGIN CommandList END
| Tableexpr
| Value
| Valuelist

Tableexpr:
  Tableexpr_c
| Tableexpr_file

Tableexpr_c:
  TABLE
| TABLEH
```

```
| XMLTABLE
| MENTTABLE
| JSONTABLE
| CSVTABLE
| HSQTABLE
| HSQTABLEH
| TXTTABLE
| RAM

Tableexpr_file:
  TAB
| XML
| TABH
| CSV
| HSQ
| HSQH
| TXT
| MENT
| JSON
| FREL
| QRY
| JPG

Strich:
  STRICH
| Strich STRICH

Bars:
  OR
| Bars OR

Value:
  NINTEGER
| INTEGER
| FLOAT
| ONR2
| RATIOTYPE
| PI
| EULER
| SEPL
| STRING
| STRING2
| STRING3
| SI
| NO

Valuelist:
  Value Valuelist
| Value Value

Operfolge:
  Oper1
| Oper1 Operfolge

Oper1:
```

```
  SUM
| ALL
| EX
| MAX
| MIN
| COUNT
| ZAHLTRIP
| COUNTSTROKE
| PROD
| ZAHLRATIO
| AVG
| DIVDIV
| MINUSMINUS
| VARIANCE
| MAD
| MEDIAN
| LINREG
| TRANSPOSE
| PLUSPLUSTEXT
| TEXT
| WORT
| ONR
| UPPER
| LOWER
| TRIM
| ZAHL
| SATZL
| LETTERL
| WORTL
| WORTM
| PZAHL
| INVERSMAT
| DET
| PERMUTATIONS
| NUMMER
| ZAHL1DE
| PZAHL1DE
| APO3
| APO4
| NORMT
| NORM10E
| NORM10M
| STANDARD
| ABS
| UNTAG0
| TOANY
| MINUSOP1
| ROUTE
| NATSEL
| NATJOIN
| SQRT
| SIN
| COS
| TAN
| ARCTAN
```

```
| LN
| RATIO
| NOT
| CREATE
| PRIMO
| ULTIMO
| RGB
| REONR
| UNTAGALL
| BARL

Oper2_all:
  Oper2
| Oper2_lt
| Oper2_or

Oper2:
  Oper2_union
| Oper2_plus
| Oper2_textsep
| Oper2_mal
| NTH
| NTHZAHL
| RAT
| MINUS
| MULTMAT
| MULTKOMPLEX

Oper2_mal:
  DDOT
| MAL
| PLUSPROZENT
| PROZENT
| MINUSPROZENT

Oper2_or:
  OR
| AND
| IMPLI
| EQUI

Oper2_union:
  UNION
| EXCEPT
| INTERSECT
| CART
| KEYS
| KEYSLIKE
| ROUND
| MANT
| LISTS
| VLISTS
| WEGE
| TREE
| ROTATE
```

```
| POLY
| TEXTEND
| TEXTINDEX
| LOAD
| SPLIT
| CUT
| CUTSPLIT
| SPLITSEP
| DCOMMA
| COMMA

Oper3:
  SUBTEXT
| SUBTEXT2
| MINUSPLUS
| Oper3_zufall
| INSIDE
| FALLS
| WEGECYC


Oper3_zufall:
  DDDOT
| DDOTX

Oper2_lt:
  LT
| GT
| LE
| GE
| NE
| EQ
| LT2
| GT2
| LE2
| GE2
| EQ2
| NE2
| IN
| INMATH
| ONEIN
| ONEINMATH
| LIKE
| SEMI

Oper2_plus:
  PLUS
| MULT
| DIV
| MOD
| DIVREST
| DIVBIGINT
| LOG
| BE
```

```
Oper2_textsep:
  PLUSPLUSTEXT2

Oper2_name:
  TAG0
| TAGALL
| COMP
| HORI

Oper2_names:
  UNTAG

Oper2_ram:
  INSERT
| UPDATE
| DELETE

Oper2_scheme:
  Oper2_scheme_standalone
| GIB

Oper2_scheme_standalone:
  GIBALL
| GIBTOP

Oper3_scheme:
  VERTI

Oper3_onrs:
  ONRS

%%
```

22 Appendix C: List of o++o color names

"aliceblue",(0.941176470588,0.972549019608,1.);
"antiquewhite",(0.980392156863,0.921568627451,0.843137254902);
"aquamarine",(0.498039215686,1.,0.83137254902);
"azure",(0.941176470588,1.,1.);
"beige",(0.960784313725,0.960784313725,0.862745098039);
"bisque",(1.,0.894117647059,0.76862745098);
"black",(0.,0.,0.);
"blanchedalmond",(1.,0.921568627451,0.803921568627);
"blue",(0.,0.,1.);
"blueviolet",(0.541176470588,0.16862745098,0.886274509804);
"brown",(0.647058823529,0.164705882353,0.164705882353);
"burlywood",(0.870588235294,0.721568627451,0.529411764706);
"cadetblue",(0.372549019608,0.619607843137,0.627450980392);
"chartreuse",(0.498039215686,1.,0.);

"chocolate",(0.823529411765,0.411764705882,0.117647058824);
"coral",(1.,0.498039215686,0.313725490196);
"cornflowerblue",(0.392156862745,0.58431372549,0.929411764706);
"cornsilk",(1.,0.972549019608,0.862745098039);
"cyan",(0.,1.,1.);
"darkgoldenrod",(0.721568627451,0.525490196078,0.043137254902);
"darkgreen",(0.,0.392156862745,0.);
"darkkhaki",(0.741176470588,0.717647058824,0.419607843137);
"darkolivegreen",(0.333333333333,0.419607843137,0.18431372549);
"darkorange",(1.,0.549019607843,0.);
"darkorchid",(0.6,0.196078431373,0.8);
"darkred",(0.5450,0.,0.);
"darksalmon",(0.913725490196,0.588235294118,0.478431372549);
"darkseagreen",(0.560784313725,0.737254901961,0.560784313725);
"darkslateblue",(0.282352941176,0.239215686275,0.545098039216);
"darkslategray",(0.18431372549,0.309803921569,0.309803921569);
"darkturquoise",(0.,0.807843137255,0.819607843137);
"darkviole",(0.580392156863,0.,0.827450980392);
"deeppink",(1.,0.078431372549,0.576470588235);
"deepskyblue",(0.,0.749019607843,1.);
"dimgrey",(0.411764705882,0.411764705882,0.411764705882);
"dodgerblue",(0.117647058824,0.564705882353,1.);
"firebrick",(0.698039215686,0.133333333333,0.133333333333);
"floralwhite",(1.,0.980392156863,0.941176470588);
"forestgreen",(0.133333333333,0.545098039216,0.133333333333);
"gainsboro",(0.862745098039,0.862745098039,0.862745098039);
"ghostwhite",(0.972549019608,0.972549019608,1.);
"gold",(1.,0.843137254902,0.);
"goldenrod",(0.854901960784,0.647058823529,0.125490196078);
"green",(0.,1.,0.);
"greenyellow",(0.678431372549,1.,0.18431372549);
"grey",(0.745098039216,0.745098039216,0.745098039216);
"honeydew",(0.941176470588,1.,0.941176470588);
"hotpink",(1.,0.411764705882,0.705882352941);
"indianred",(0.803921568627,0.360784313725,0.360784313725);
"ivory",(1.,1.,0.941176470588);
"lavender",(0.901960784314,0.901960784314,0.980392156863);
"lavenderblush",(1.,0.941176470588,0.960784313725);
"lawngreen",(0.486274509804,0.988235294118,0.);
"lemonchiffon",(1.,0.980392156863,0.803921568627);
"lightblue",(0.678431372549,0.847058823529,0.901960784314);
"lightcoral",(0.941176470588,0.501960784314,0.501960784314);
"lightcyan",(0.878431372549,1.,1.);
"lightgoldenrod",(0.933333333333,0.866666666667,0.509803921569);
"lightgray",(0.827450980392,0.827450980392,0.827450980392);
"lightpink",(1.,0.713725490196,0.756862745098);
"lightsalmon",(1.,0.627450980392,0.478431372549);
"lightseagreen",(0.125490196078,0.698039215686,0.666666666667);
"lightskyblue",(0.529411764706,0.807843137255,0.980392156863);
"lightslateblue",(0.517647058824,0.439215686275,1.);
"lightslategray",(0.466666666667,0.533333333333,0.6);
"lightsteelblue",(0.690196078431,0.76862745098,0.870588235294);
"lightyellow",(1.,1.,0.878431372549);

"limegreen",(0.196078431373,0.803921568627,0.196078431373);
"linen",(0.980392156863,0.941176470588,0.901960784314);
"ltgoldenrodyello",(0.980392156863,0.980392156863,0.823529411765);
"magenta",(1.,0.,1.);
"maroon",(0.690196078431,0.188235294118,0.376470588235);
"mediumaquamarine",(0.4,0.803921568627,0.666666666667);
"mediumblue",(0.,0.,0.803921568627);
"mediumorchid",(0.729411764706,0.333333333333,0.827450980392);
"mediumpurple",(0.576470588235,0.439215686275,0.858823529412);
"mediumseagreen",(0.235294117647,0.701960784314,0.443137254902);
"mediumslateblue",(0.482352941176,0.407843137255,0.933333333333);
"mediumturquoise",(0.282352941176,0.819607843137,0.8);
"mediumvioletred",(0.780392156863,0.0823529411765,0.521568627451);
"medspringgreen",(0.,0.980392156863,0.603921568627);
"midnightblue",(0.0980392156863,0.0980392156863,0.439215686275);
"mintcream",(0.960784313725,1.,0.980392156863);
"mistyrose",(1.,0.894117647059,0.882352941176);
"moccasin",(1.,0.894117647059,0.709803921569);
"navajowhite",(1.,0.870588235294,0.678431372549);
"navyblue",(0.,0.,0.501960784314);
"oldlace",(0.992156862745,0.960784313725,0.901960784314);
"olivedrab",(0.419607843137,0.556862745098,0.137254901961);
"orange",(1.,0.647058823529,0.);
"orangered",(1.,0.270588235294,0.);
"orchid",(0.854901960784,0.439215686275,0.839215686275);
"palegoldenrod",(0.933333333333,0.909803921569,0.666666666667);
"palegreen",(0.596078431373,0.98431372549,0.596078431373);
"paleturquoise",(0.686274509804,0.933333333333,0.933333333333);
"palevioletred",(0.858823529412,0.439215686275,0.576470588235);
"papayawhip",(1.,0.937254901961,0.835294117647);
"peachpuff",(1.,0.854901960784,0.725490196078);
"peru",(0.803921568627,0.521568627451,0.247058823529);
"pink",(1.,0.752941176471,0.796078431373);
"plum",(0.866666666667,0.627450980392,0.866666666667);
"powderblue",(0.690196078431,0.878431372549,0.901960784314);
"purple",(0.627450980392,0.125490196078,0.941176470588);
"red",(1.,0.,0.);
"rosybrown",(0.737254901961,0.560784313725,0.560784313725);
"royalblue",(0.254901960784,0.411764705882,0.882352941176);
"saddlebrown",(0.545098039216,0.270588235294,0.0745098039216);
"salmon",(0.980392156863,0.501960784314,0.447058823529);
"sandybrown",(0.956862745098,0.643137254902,0.376470588235);
"seagreen",(0.180392156863,0.545098039216,0.341176470588);
"seashell",(1.,0.960784313725,0.933333333333);
"sienna",(0.627450980392,0.321568627451,0.176470588235);
"silver",(0.898039215686, 0.898039215686, 0.898039215686);
"skyblue",(0.529411764706,0.807843137255,0.921568627451);
"slateblue",(0.41568627451,0.352941176471,0.803921568627);
"slategrey",(0.439215686275,0.501960784314,0.564705882353);
"snow",(1.,0.980392156863,0.980392156863);
"springgreen",(0.,1.,0.498039215686);
"steelblue",(0.274509803922,0.509803921569,0.705882352941);
"tan",(0.823529411765,0.705882352941,0.549019607843);

```
"thistle",(0.847058823529,0.749019607843,0.847058823529);
"tomato",(1.,0.388235294118,0.278431372549);
"turquoise",(0.250980392157,0.878431372549,0.81568627451);
"violet",(0.933333333333,0.509803921569,0.933333333333);
"violetred",(0.81568627451,0.125490196078,0.564705882353);
"wheat",(0.960784313725,0.870588235294,0.701960784314);
"white",(1.,1.,1.);
"whitesmoke",(0.960784313725,0.960784313725,0.960784313725);
"yellow",(1.,1.,0.);
"yellowgreen",(0.603921568627,0.803921568627,0.196078431373)
```