o++o on 17 Pages

(02.04.2020)

Klaus Benecke

Facebook and Co. are working on preparing messages in such a way that they appeal even more emotionally, as if the social situation is not already heated up enough. We are working to provide end users with tools to strengthen their rational judgment using the computer. For this you need programming languages that are as simple as possible, yet powerful, and extensive, trustworthy, publicly accessible information in the form of large tables and documents similar to Wikipedia.
Even if the developed language is as simple as possible, in contrast to the "Facebook approach" it will require a certain amount of learning.
Such a programming language in combination with trustworthy data could be a big step towards further democratization of society. Many false messages could easily be paralyzed by anyone by means of relevant facts or statistical evaluations.
Perhaps one can compare the creation of such a programming language with the creation of the first alphabet by the Phoenicians or the creation of the first alphabet with vowels by the Greeks. These peoples could have done such things without these prerequisites. I believe there would have been no Greek science and culture without this alphabet; maybe not a Greek democracy either.

Design criteria for such a language:

1. A mathematical foundation is required.
2. Methodological-didactic and pragmatic questions initially face efficiency problems.
3. Short, readable programs; the main keywords should be short
4. Simple, unstructured programs; Loops often lead to programs that are difficult to read and difficult to change; Recursion requires a relatively high level of abstraction
5. Universal applicability; it must not only be usable for relations (flat simple tables) but also for structured tables and documents; it must not only be suitable for queries to the most important systems, but also for a variety of calculations
6. In order to be used in school lessons, it must support the various mathematical sub-areas and offer benefits for the other subjects
7. It should be powerful enough to replace other systems and languages like spreadsheets and SQL.
8. From the end-user perspective, there can only be one uniform system with uniform syntax (spelling) for the processing of mass data, just as the operations for single data processing (+ - *: sqrt sin ...) are standardized.

o++o, more precisely ottoPS (otto programming language), is an improved pocket calculator that also allows queries to structured TABles and docuMENTs (tabments). o++o is also suitable for enabling database queries and later also being the basis of a search engine.
o++o was designed so that the programs are as short as possible, with simple and clear syntax. The following programs should be tried out at ottoPS.de, because you can hardly learn programming without your own testing.

Program 1: Calculate the fourth root from 2.

```
2 sqrt sqrt
```

You can see that unary operations are written according to the input value (postfix). This saves us 4 brackets compared to the well-known spelling sqrt (sqrt (2)).

Program 2: Calculate the sine of 1 (radians).

```
1 sin
```

Program 2b: Compute the sine of 30°.
```
30:180*pi sin
```

Program 3: How many 10 digit dual numbers are there?
```
2 hoch 10
```
`hoch` is German and means `„to the power of“`

Program 4: Calculate the edge length of a cube of volume 2 (the third root of 2).
```
2 hoch (1:3)
```
or
```
2 hoch 1/3
```
1/3 is a rational number. That is, "/" is not an operation in contrast to ":". Since we always calculate from left to right, 1 to the power 1: 3 = (1 to the power of 1): 3 = 0.333333333.

Program 5: Compute an average.
```
1 3 2 1 3 4 ++:
```
In order to save writing effort, with a list of values (here the numbers of the line) there is no visible separator and no brackets. The operation of averaging (++:) is now applied to this list. Instead of the average, one can also form a sum (++), the product (**), count (++1), the maximum (max), the sine (sin), ln, ... Since *sin* only requires one input value, a list of output values results when using the sine operation instead of the ++: operation. You could then use ++: on this list and only get a number. The above spelling is more compact than the old spelling
++: ([1 3 2 1 3 4])
In particular, if several operations are used in succession, this saves many brackets and thus causes of errors. We consider ++: and the other operations mentioned here to be single-digit (unary), just like sin or sqrt, and write them according to the input value, since we consider the given numbers as a list (a tabment).Um Schreibaufwand zu sparen, kann man bei einer Liste von Werten (hier die Zahlen der Zeile) auf jegliches sichtbare Trennzeichen und jegliche Klammerung verzichten. Auf diese Liste wird jetzt die Operation der Durchschnittsbildung (++:) angewandt. Man kann anstelle des Durchschnittes auch eine Summe (`++`), das Produkt (`**`), die Anzahl (`++1`), das Maximum (`max`), den Sinus (`sin`), ln, … bilden. Da `sin` nur einen Inputwert benötigt, ergibt sich bei Anwendung der Sinusoperation anstelle der ++: -Operation eine Liste von output-Werten. Auf diese Liste könnte man dann wieder ++: anwenden und erhält dann lediglich eine Zahl. Obige Schreibweise ist zunächst vielleicht etwas gewöhnungsbedürftig, aber sie ist kompakter als die alte Schreibweise
```
++:([1 3 2 1 3 4])
```
In particular, if several operations are used in succession, this saves many brackets and thus causes of errors. We consider ++: and the other operations mentioned here to be single-digit (unary), just like sin or sqrt, and write them according to the input value, since we consider the given numbers as a list (a tabment).

Program 6: Compute the valueof the term „`sqrt(abs(sin(7.1))+abs(cos(8.1)))“.`
```
7.1 sin abs
+ 8.1 cos abs
sqrt
```
or in one line
```
7.1 sin abs + (8.1 cos abs) sqrt
```
In the three-line solution, as usual in programming languages, the calculation is from top to bottom. You cannot calculate the value of the second line if you do not omit the "+" sign. Therefore, the

values of the first and second lines are simply added up. The root of the overall result of the second line is then drawn through the third line.

Program 7: Subtract 5 more numbers from the first number.

```
500 77.9 45.6 33.5 23.74 25.88 --
```

In o++o several operations are written according to the very old "forest principle". There is a word for the tree and TreeTree means forest. The above -- therefore replaces 5 minus signs.

Program 8: Multiply each decimal number in a list by another number.

```
2.40 2.70 7.90 * 1.19
```

Each number in the input list is multiplied by 1.19. If the numbers given are net prices, the result represents the corresponding gross prices; if the given numbers are amounts in one currency and 1.19 is the conversion rate, the result shows the values in the other currency, if the numbers in the list are lengths of rectangles, then there are 3 areas of rectangles, ... We see that decimal numbers are not represented with the comma but with the dot.

Program 9: Form the sum of many positive and many negative numbers without using many minus signs and brackets.

```
4 5 3 2 1  8 9 ++
- 7 6 5 4 3 2 1 ++
```

Program 10: Calculate the circumference of several circles, given the radii.

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

Program 11: Calculate the circumferences and areas of several circles, given the radii.

```
[R! 4 5 6 2 3.7 9.77]
CIRCUMFERENCE:=R*pi*2
AREA:=R*R*pi
```

The R gives each element of the given list the name (tag) R. By the assignment (:=) the given table is expanded by a new column. The `CIRCUMFERENCE` and `AREA` columns are added one after the other at the top, so that a table of type `R,CIRCUMFERENCE,AREA l` results. l stands for list.

Program 12: Berechne die Flächen und Umfänge mehrerer Rechtecke.

```
<TAB!
A,   B l
1.23 5.67
7.65 4.32
9.87 6.54
!TAB>
CIRCUMFERENCE:=A+B*2
AREA:=A*B
```

The TAB brackets (<TAB!,! TAB>) are only used in the program part of the system. In the TAB display, the values must start exactly under the associated column names. In the case of small input tables, a single-line notation can also be used.

```
[A,B! 1.23 5.67 7.65 4.32 9.87 6.54]
CIRCUMFERENCE:=A+B*2
AREA:=A*B
```

A table with 4 columns is created.

Program 13: Determine the total price of a simple invoice.

```
<TAB!
ARTICLE, PRICE l
Beer     0.61
Limo     0.23
Steak    2.40
!TAB>
++
```

Here the sum of all the numbers in the given table (list (l) of pairs) is simply formed. The article values are words and therefore have no influence on the result. Replace ++ with

```
* 1.10
```

, a table with 2 columns and three lines (sentences, tuples) is created again, each number including the tip. Then you can again

```
++
```

append to get the total.

Program 14: Bestimme den Gesamtpreis einer etwas komplizierteren Rechnung, die durch eine einfache Tabelle gegeben ist.

```
<TAB!
ARTICLE, PRICE, NO l
Beer     0.61   7
Limo     0.23   3
Steak    2.40   4
!TAB>
POSPRICE:=PRICE*NO
++ POSPRICE
```

The assignment extends the given table by a new column with the column name POSPRICE, whereby the price is multiplied three times by the corresponding number. In order to determine the total amount of the pub visit, you have to give the ++ operation a second input value. Otherwise the total of all 9 numbers in the intermediate table would be formed. Both program lines can also be simply repaced by:

```
++ PRICE*NO
```

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

The symbol := of the assignment is to be distinguished from the equal sign =. The equal sign as well as <,>, <=, in, ... is required to formulate conditions. Conditions are used for selection. For example, if you add a condition

```
avec ARTICLE=Beer
```

or simply

```
avec Beer
```

, the end result is only the total price for the 7 beers. If you want to calculate the price for the rest, you can instead insert

```
sans ARTICLE=Beer
```

or simply

```
sans Beer
```

insert. Column names (metadata) must always be capitalized. The keywords (gib, sans, avec, ...) must always be written in lower case. If you always write a word of the primary data with upper and lower case letters, the program becomes easier to read.

Program 15: Determine the total price of a long pub visit.

```
<TAB!
ARTICLE,PRICE,NOl l
Beer    0.61  7 6 5
              3
Limo    0.23  3 4
Steak   2.40  4 3 2
!TAB>
POSPRICE:=PRICE*NO
++ POSPRICE
```

We are dealing with a structured table that contains several orders for each item. We do not have to write the NO entries one below the other so that you can represent the visit to the pub in a compact manner. If, on the other hand, you do not look at the end result but only the result of applying the assignment, the NO and POSPRICE values must be underneath each other, since the resulting structured table would otherwise become too confusing:

```
ARTICLE, PRICE, (NO, POSPRICE l)l
Bier     0.61    7   4.27
                 6   3.66
                 5   3.05
                 3   1.83
Limo     0.23    3   0.69
                 4   0.92
Steak    2.4     4   9.6
                 3   7.2
                 2   4.8
```

If you first want to add the NO values and then multiply them,

```
POSPRICE:= NOl ++ *PRICE
```

, then results a more compact intermediate table:

```
ARTICLE, PRICE, POSPRICE,NOl l
Beer     0.61   12.81    7 6 5 3
Limo     0.23    1.61    3 4
Steak    2.4    21.6     4 3 2
```

Both program lines can also be replaced by ++ PRICE * NO.

You can save the tables in question under a name, for example with the ending .tab, and then call them up again. This means that tables or documents can be used in several programs.

Program 15 could then look like:

```
pubvisit.tab
++ PRICE*NO
```

This example can be applied to many applications. When bowling you could also build a table `bowling.tab` with repeating group: NAME, THROWl m. m truncates set and l list. For each bowler you can then determine the total by an assignment and then sort the data downhill. If you only want certain columns in the result and you still want to sort them according to these, you need a gib-clause.

Program 16: Determine the overall result of bowling for each person and then sort the data.

```
bowling.tab
TOTAL:=THROWl ++
gib TOTAL,NAME m-
```

It is also a little shorter:

Program 17: See program 16.

```
bowling.tab
gib TOTAL,NAME m-
    TOTAL:= THROW ! ++
```

Here the kind of the aggregation (++) results from the head of the desired table. TOTAL is one aggregation per NAME. For sets (m, m-), they are always sorted according to the column names specified first.
We assume that every river in `rivers.tab` has a LENGTH column. Then you can find all rivers that are longer than 500 km by the following query:

Program 18: Select in a given table.

```
rivers.tab
avec LENGTH>500
```

Now the rivers are still to be sorted according to the length and are issued only with tributaries:

Program 19: Selection with subsequent sorting.

```
rivers.tab
avec LENGTH > 500
gib LENGTH, RIVER, TRIBUTARYm m
```

If you want the longest rivers first, you just have to replace the outer m by m-.
Our file rivers.tab also contains a repeating group for each river `COUNTRY,BUNDESLANDm m`, which indicates by which countries and federal states the river flows. In rivers.tab, therefore, the RIVER is superior to the federal state. If you want to reverse this, it can be realized again simply by specifying the header of the desired table.

Program 20: Restructuring of the rivers-file.

```
aus rivers.tab
gib BUNDESLAND,RIVERm m
```

Through this free reorganization, all rivers that flow through it are collected for each federal state. The federal states and the rivers within the federal states are sorted. A river can occur here in several states. Each federal state appears only once, because we have chosen a set as outer collection. The beginning of a program can also be marked by `aus`.
In addition to the simple calculations (: =), which extend a table by a new column, there are the so-called recursive extensions. Here are two formulas given. The first formula is for the first element of the collection and the second for the remaining elements. The formula for the remaining elements can refer to the predecessor of the new column. This is in the following program `AMOUNT pred`.
Program 21: What will become of 100 euros after 20 years at 9% growth (interest).

```
[YEAR! 0 .. 20]
AMOUNT:= first 100. next AMOUNT pred*1.09 at YEAR
```

Program 22: Convert a binary number (here 10111 = 23) to a decimal number.

```
[BIT! 1 0 1 1 1]
DECI:= first BIT next DECI pred*2+BIT at BIT
```

If not the whole development but only the last element of the list is desired, then one can add the condition `avec BIT pos- = 1`. If the last bit is not desired, you should follow `gib DECI`.
Program 23. Convert a decimal number (here 23) to a binary number.

```
DIV,REST l:= first 23 divrest 2
             next  DIV pred divrest 2
             while DIV,REST != 0,0
```

```
gib RESTl-
```
With the above recursive extension, 2 new columns are introduced, DIV and REST. `divrest` is here the integer division with the integer remainder (a pair of numbers).

Program 24: Calculate the (German) weighted average for 3 students and the overall average.
```
<TAB!
NAME,  EXAMl,MARKl l
Ernst  1 2   1 2 3 1 3 1 1
Clara  1 1   3 1 2
Sophia 1 3   1 1 2
!TAB>
AVG:=EXAMl ++: *0.6 +(MARKl ++: *0.4)
TOTAL:=AVGl ++:
rnd 2
```
The result is rounded by `rnd`.

Program 25: Make a union (all StudentIDs included in one of the tables).
```
aus exams.tab, projects.tab
gib STIDm
```
Here are `exams.tab` and `projects.tab` of type `STID,COURSE,MARK m` resp. `STID,PROJ,HOURS m`.

Program 26: Make an intersection (all StudentIDs included in both tables).
```
aus  exams.tab, projects.tab
igib STIDm
```
With `igib` symultaneously joins can be expressed and thus also queries to entire databases are formulated. The following queries refer to a database `uni.tab` whose second table is unstructured. The queries do not need to be modified or only weakly modified if all `uni.tab` tables are unstructured (relational).
```
<STUDENTS!
STID, NAME,  LOC?,    STIP, FAC,  (COURSE,     MARK m),(PROJ, HOURS m) m
1234  Ernst  Oehna    500   Mathe  Algebra     1        Fritz  4
                                  History     1        Otto   2
                                  Logic       2
1245  Sophia Berlin   400   CS     Algebra     3        Deng   5
                                  Databases   1        Ming   4
                                  Otto        1        Otto   6
3456  Clara  Oehna    450   CS     Databases   1
                                  OCaml       2
4567  Ulrike          400   Art                         Monet 10
5678  Kaethe Gerwisch   0   Art    Apel        1        Monet 20
                                  Repin       1
!STUDENTS>
<FACS!
FAC,  DEAN,   BUDGET, STUDKAP m
CS    Reichel 10000   500
Art   Sitte   2000    600
Maths Gauss   1000    200
Philo Hegel   1000    10
!FACS>
```
The STUDENTS table is structured because 7 components are stored for each set (structured tuple) (students), and the last two components themselves are small tables. For example, the penultimate component of Clara contains two exam results and the last one the empty set of projects.

Program 27: Calculate the total budget of the university.

```
uni.tab
++  BUDGET
```

Program 28: Calculate the total budget and the average budget of the university.

```
aus uni.tab
gib SUMBUD,AVGBUD  SUMBUD:= BUDGET ! ++
                   AVGBUD:= BUDGET ! ++:
```

Program 29: How many students and faculties does the university have.

```
uni.tab
++1
```

Program 30: For each course, give the associated students a grade.

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

Program 31: Give all records (tuples) that contain 1234.

```
aus  uni.tab
avec 1234
```

Program 32: In all tables containing the student identifier STID, select the student data with the number 1234.

```
uni.tab
avec STID=1234
```

Program 32 differs only with respect to the FAKS table from program 31. In the result of program 31 this is empty and in program 32 it is completely preserved. In other applications, instead of the student identifier (STID), you could also select article numbers or part numbers.

Program 33: Give the dean and his exams to the student with the number 1234.

```
aus  uni.tab
avec STID=1234
igib NAME,DEAN,(COURSE,MARK m)m
```

Here a "join" without join condition is realized. If you were to use `gib` instead of i`gib`, the result set would be left empty because there is no connection between NAME and DEAN.

Program 34: Calculate the number of ones for each faculty.

```
aus  uni.tab
avec MARK=1
gib  FAC,COUNT m  COUNT:= MARK ! ++1
```

Program 35: Test the following program.

```
aus  uni.tab
avec DEAN in [Reichel Dassow]
gib  FAC,DEAN,(NAME,STIP m)m
```

Program 36: Give all the students to the faculties of Gauss and Reichel.

```
uni.tab
avec DEAN in [Reichel Gauss]
igib FAC,DEAN,(NAME,STIP m)m
```

Program 37: Wanted are all faculties that have students with the marks 1, 1 and 3.

```
aus  uni.tab
avec {{1 1 3}} inm MARKb  # inm: in mathematical
igib FAC, DEAN m
```

Program 38: We are looking for all students who have exactly 2 ones and only these.

```
aus  uni.tab
avec MARKl = [1 1]
gib STUDENTS
```

Program 39: We are looking for all data from Ernst (including his faculty data).

```
aus  uni.tab
avec NAME=Ernst
igib
```

The program does not need to be changed if all tables of `uni.tab` are in the first normal form (unstructured).

Program 40: A bottle and a cork costs one mark and ten. The bottle is a mark more expensive than the cork. How much is the cork?

```
[BOTTLE! 0 .. 110]
CORK:=BOTTLE - 100
avec CORK+BOTTLE = 110
```

Program 41: Write 1000 times *I love you*.

1000 mal "I love you!"

Program 42: Compute the total number of occurrences of all assemblies and all individual parts of the Hyundai I40 with aluminum rim and without air conditioning.

```
<TAB!
SUP,          PROPERTY,  (SUB,           OCC l)m
bushing       cylindrical
rim                      aluminum rim    1
                         steel rim       1
i30           modern     wheel           4
                         engine          1
                         body            1
i40           fast       wheel           4
                         spare           1
                         body            1
                         conditioning    1
                         engine          1
body          blue
conditioning  clean
piston ring   round
piston        light      piston ring     2
                         bushing         1
engine        heavy      piston          6
                         screw           8
wheel         round      screw           5
```

```
                          tire           1
                          rim            1
tire          black
screw         stable
!TAB>
avec SUB! SUP= rim -> SUB="aluminum rim"
sans SUB! SUP = i40 & SUB=conditioning     # Optional condition
wege i40      # all paths starting in i40 are generated
AUX:=first OCC next AUX pred*OCC at OCC
avec SUB! SUB pos- =1
gib SUB,TOTAL m
    TOTAL:= AUX ! ++
```

result:

```
SUB,          TOTAL  m
aluminum rim   4
body           1
bushing        6
engine         1
piston         6
piston ring   12
rim            4
screw         28
spare          1
tire           4
wheel          4
```

Program 43: Determine the total price of a long visit to the pub, using a separate price table.

```
<TAB!
ARTICLE,NOl m
Beer    2 4 5
Limo    3 2
Steak   4 3
!TAB>
, articles.tab
igib TOT,(ARTICLE,TOT m)  TOT:= NO*PRICE ! ++
```

Ergebnis:

```
TOT0, (ARTICLE,TOT1  m)
79.3   Beer    29.7
       Limo    16.
       Steak   33.6
```

articles.tab:

```
ARTICLE,PRICE  m
Beer    2.7
Limo    3.2
Steak   4.8
Cake    4.8
```

Program 44: We are looking for the Dean of Ernst. (query to a Relational database.)

```
unirelational.tab
avec NAME=Ernst
igib DEKAN
```
unirelational besitzt hierbei das folgende Schema:
```
TABMENT! UNIRELATIONAL
UNIRELATIONAL! STUDENTEN,EXAMEN,PROJEKTE,FAKS
PROJEKTE! (STID,PROJ,STUNDEN m)
STUDENTEN! (STID,NAME,ORT?,STIP,FAK m)
EXAMEN! (STID,KURS,NOTE m)
FAKS! (FAK,DEKAN,BUDGET,STUDKAP m)
```

Program 45: Give me all the data from Ernst.
```
unirelational.tab
avec NAME=Ernst
igib
```
Result as hsq-table:
```
STID NAME LOC STIP FAC
 STID COURSE MARK
  STID PROJ HOURS
   FAC DEAN BUDGET STUDKAP
1234 Ernst Oehna 500 Mathe
 1234 Algebra 1
 1234 Geschichte 1
 1234 Logik 2
  1234 Fritz 4
  1234 Otto 2
   Mathe Dassow 1000 200
```
It should be noted that all of Ernst's data appears in the result without using a cross product. Otherwise, the “Fritz” and “Otto” projects would appear 3 times each.

Program 46: Exam results are sought for student number 1234.
```
unirelational.tab
avec STID=1234
igib NAME,DEAN,(COURSE,MARK m)m
```
Result:
```
NAME, DEAN,  (COURSE,  MARK  m) m
Ernst Dassow  Algebra  1
              History  1
              Logic    2
```
It should be noted that the result contains information from 3 tables without using a join condition.

o++o for School

A. Merkel "In addition to reading, calculating and writing, every student should also learn programming." o++o (detailed ottoPS) is a table-oriented programming language that does not use loops. Nevertheless, o++o is very expressive and you can use it not only to perform compact

queries but also to perform a wide range of calculations for structured tables and structured documents.
o++o uses and conveys many mathematical concepts, so we see the main advantages of teaching in math classes, as well as the essential skills for using the calculator in mathematics. o++o uses the following concepts in particular: set, multi-set, list, equality and inclusion relationships of these; Tuple; powerful operations for selection; calculating; restructuring, sorting and aggregation (sum; average; ...), ...
Spreadsheet programs like EXCEL and the database standard query language SQL do not have any structured schemes and tables. Initial tests with preschool children suggest that structured tables are easier to calculate than decimal numbers. We want to add more o++o sample programs:

Program 47. Calculate the value of a simple term.
```
2*3+4
```
* and + each have 2 input values. First, 2 * 3 (6) is calculated. The 6 is the first input value of +, so that there is a total of 24. So here is simply calculated from left to right.

Program 48. Write the term $\cos^3(\sin^2(3.14159))$ in o++o.
```
pi sin hoch 2 cos hoch 3
```
In our opinion, the starting term is difficult to read for Otto normal consumers . You start with `pi`, go left to `sin`, then right to `high 2`, now move left to cos and finally right to `high 3`. This spelling has been introduced to save parentheses. Actually, to be unmistakable, the starting date should look like this:
$(\cos((\sin(3.14159))^2))^3$
This is certainly more difficult to read and you move even more from left to right and vice versa.

Program 49. Write the term sin2 (x) + cos3 (y) in o++o.
```
X sin hoch 2 + (Y cos hoch 3)
```
or
```
X sin hoch 2
+ Y cos hoch 3
```
You could write all terms in o++o without parentheses, but then certain terms would have to be written in multiple lines.

Program 50. How do you calculate the term 2 + 3: 4 * 5?
```
2+(3:(4*5))=2 3/20
2+((3:4)*5)=5 ¾
o++o: ((2+3):4)*5=6 1/4
```
You can see that school wisdom points calculation does not get by before line calculation. You also need the rule "from left to right".

Program 51. Calculate the average of several grades.
```
1 2 3 1 2 ++:
```
This program can be improved from a methodological point of view by adding the brackets for lists:
```
[1 2 3 1 2] ++:
```
It can now be seen that the average operation ++: has one input value, namely a list, and that ++: is followed to the input value. Since users generally do not want to type much, we assume that the first notation will be used more frequently in practice.

Program 52. Calculate the averages of the structured table marks.tab for each subject.
```
marks.tab
AVG:= MARKl ++:
```

marks.tab could look like:

```
SUBJECT,MARKl l
Ma      1 2 1 3 1 2
Phy     4 3 2 2 1
```

Here, the list is abbreviated by l. That is, marks.tab is a simple structured table (list) that contains a list of marks for each subject. To save space, we also choose the methodologically not optimal presentation. Like SUBJECT, MARK is a column name, so marks.tab should actually be represented like this:

```
SUBJECT,MARKl l
Ma      1
        2
        1
        3
        1
        2
Phy     4
        3
        2
        2
        1
```

The result of the query again in "tab-format":

```
SUBJECT, AVG,           MARKl  l
Ma       1.66666666667 1 2 1 3 1 2
Phy      2.4           4 3 2 2 1
```

Program 53. Form the sum of the numbers from 1 to 100 (task of Gauss class 5).

```
1 .. 100 ++
```

Like addition and multiplication, ".." has two input values (1 and 100). The following list is the intermediate result:

```
ZAHLl
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

It resuts `5050.`

Program 54. Approximately calculate the maximum of the sine function in the interval [1 2].

```
1 ... 2!0.001 sin max
```

"..." requires 3 input values: 1. the start value 1, the end value 2 and the step size 0.001. The numbers 1 1,001 1,002 1,003… 1,999 2 result.

The sine function is applied to each of the numbers, so that 1001 numbers are generated again. The function max (maximum) is then applied to this list. Although this is an approximation, the exact value of 1 comes out if the step size is further refined. `sin` and `max` each have an input value (here a list) but the output value of `sin` is again a list and `max` only generates a number, since this is an aggregation function. The second and third input values of a three-digit operation (above ...) are separated by a "!". This is necessary in o++o, since the comma has already been assigned for pair formation and the space already separates list elements.

Program 55. Approximately calculate the minimum of the polynomial $X^3 + 4 X^2 -3 X + 2$ in the interval [0 2] with the associated X value.

```
[X! 0 ... 2!0.001]
Y:= X polynom [1 4 -3 2]
MINI:= Yl min
avec Y = MINI
```

`avec` is French and denotes a selection. A concrete polynomial of a variable X always has only one input value that is used for X. In contrast, the polynomial in line 2 is more general and has 2 input values:

1. The input value for X, which here accepts all numbers that were generated in the first line
2. A list of numbers that corresponds to the coefficients of the specific polynomial.

The first line creates a list of numbers that have all been given the name X. This can be seen best in the xml or ment representation:

<X>0.</X>
<X>0.001</X>
<X>0.002</X>
…

Final result:

```
MINI,       (X,     Y         l)
1.481482037 0.333 1.481482037
```

Program 56. Approximately calculate a null of the cosine function in the interval [1 2].

```
[X! 1 ... 2!0.0001]
avec X cos < 0
avec X pos = 1
```

After the first selection, only the X values with a function value less than 0 remain here. Of these, the first value is selected in the second step. Since we know that `cos` has only one null in the interval under consideration, this is approximated by the result.

Program 57. Calculate the total growth if there are 5 annual growth numbers. Round the result to one decimal digit.

```
[W! 0 1.5 2.1 1.3 0.4 1.2]
ACCU:= first 100. next ACCU pred *(W:100+1) at W
rnd 1
```

Result:

```
W,  ACCU  l
0.  100.
1.5 101.5
2.1 103.6
1.3 105.
0.4 105.4
1.2 106.7
```

The first ACCU value results from the expression after `first` (100.). For the second value, the value `100.` is used for `ACCU pred` and the term is evaluated according to next. The result is `101.5`. This number is used again in `ACCU pred` and the next term is recalculated (around 103.6), ... until the last W value is reached. `pred` is the predecessor.

Program 58. Approximately calculate the area under the sine curve in the interval [0, pi].

```
0 ... pi!0.0001 sin *0.0001 ++
```

Here, all numbers between 0 and pi are generated one after the other, then the sine is calculated from each number and then each number is multiplied by 0.0001. 10000 rectangular areas are created, which are then added.

Program 59. Calculate the average BMI per age and the BMI per person and age for all people over 20.

```
<TAB!
NAME,       LENGTH,   (AGE, WEIGHT l) l
Klaus       1.68       18   61
                       30   65
                       56   80
Rolf        1.78       40   72
Kathi       1.70       18   55
                       40   70
Walleri     1.00        3   16
Viktoria    1.61       13   51
Bert        1.72       18   66
                       30   70
!TAB>
avec NAME! 20<AGE
BMI:= WEIGHT : LENGTH : LENGTH
gib  AGE,BMIAVG,(NAME,BMI m) m  BMIAVG:= BMI ! ++:
rnd 2
```

The TAB brackets indicate that the included data correspond to the TAB representation.
The above condition selects person records, i.e. NAME, LENGTH, (AGE, WEIGHT l) tuple (structured tuple or struple). Since a person has more than one age entry, it must be quantified. NAME! 20 <AGE therefore selects all persons who have a corresponding age entry. This means that the existence quantifier is not written, but it belongs to every condition. In this small example, the selection could of course also be carried out by hand.
Result:

```
AGE   , BMIAVG, (NAME, BMI   m) m
18      20.98     Bert  22.31
                  Kathi 19.03
                  Klaus 21.61
30      23.35     Bert  23.66
                  Klaus 23.03
40      23.47     Kathi 24.22
                  Rolf  22.72
56      28.34     Klaus 28.34
```

The result can, for example, be displayed as a bar chart by simply clicking on it. The example shows that you can reverse a hierarchy simply by specifying the desired scheme. As a result, the name is subordinate to the age.
It becomes particularly clear that the above tasks can be solved without knowledge of the differential and integral calculus. Mathematics lessons can be supported in a variety of ways with o++o. This ranges from grade 7 or lower to grade 12. It concerns: calculating with natural numbers, decimal numbers, approximate calculation of nulls of any function, derivation, areas under curves, extreme values (can already be taught in secondary school), probability calculation, ... With o++o you can easily calculate things that are otherwise only dealt with theoretically. This can significantly improve, expand and deepen the understanding of the concepts. Further information on o++o can be found at ottops.de.

We believe that o++o offers special advantages for mathematics and computer science lessons, but can also be used in other subjects (in future queries to Wikipedia).

What is scheme of a structured table?

First, a table is divided into n columns A1, A2, ..., An. The column names in the o++o program must not contain lowercase letters. If A1, A2, .. have an elementary type, an A1, A2, ..., An table contains exactly one simple line, e.g .:

```
NAME,LOC,  SALARY
Paul Oehna 1000
```

If we add a collection symbol, e.g. l for a list, the table can contain 0.1 or more rows. Example:

```
NAME,  LOC,    SALARY l
Paul   Oehna   1000
Sophia Dallgow  900
Emily  Dallgow 2000
Clara  Oehna    900
```

If we replace the l with the set symbol m, the table is shown sorted.

```
NAME,  LOC,    SALARY m
Clara  Oehna    900
Emily  Dallgow 2000
Paul   Oehna   1000
Sophia Dallgow  900
```

If you want to sort this table by location, just swap the columns in a corresponding statement
`gib LOC,NAME,SALARY m.`

```
LOC,    NAME,  SALARY  m
Dallgow Emily  2000
Dallgow Sophia  900
Oehna   Clara   900
Oehna   Paul   1000
```

Now you can see that the table contains some redundancy. You can eliminate these in a structured table:

```
LOC,   (NAME,  SALARY  m) m
Dallgow Emily  2000
        Sophia  900
Oehna   Clara   900
        Paul   1000
```

This table contains 2 structured tuples, i.e. e.g. the number (++1) of the elements in the table is no longer 4 but 2. Nevertheless, we can process this table with o++o in exactly the same way as the previous ones. You can also create 2 or 3 tables from one source table with a gib statement:

```
LOCm,   NAMEm, SALARYm
Dallgow Clara   900
Oehna   Emily  1000
        Paul   2000
        Sophia
```

Although this overall table contains all the elementary data from the given table, the additional information has been lost. Dallgow and Clara are on the same line in this TAB representation, but Clara does not live in Dallgow. Therefore, you have to look carefully at the scheme.

Ein mensch dem zahl (o++o) verborgen ist
Leichtlich der verführt wird mit list
Das nimm zu hertzen bitt ich sehr
Und jeder sein kind rechnen (programmieren) lehr ...

Adam Ries

A person who is hidden from numbers (o++o)
Easily is enticed with cunning
I ask you to take this very seriously
And every body teaches his child calculating (programming) ...