Ideen für eine APL-Library

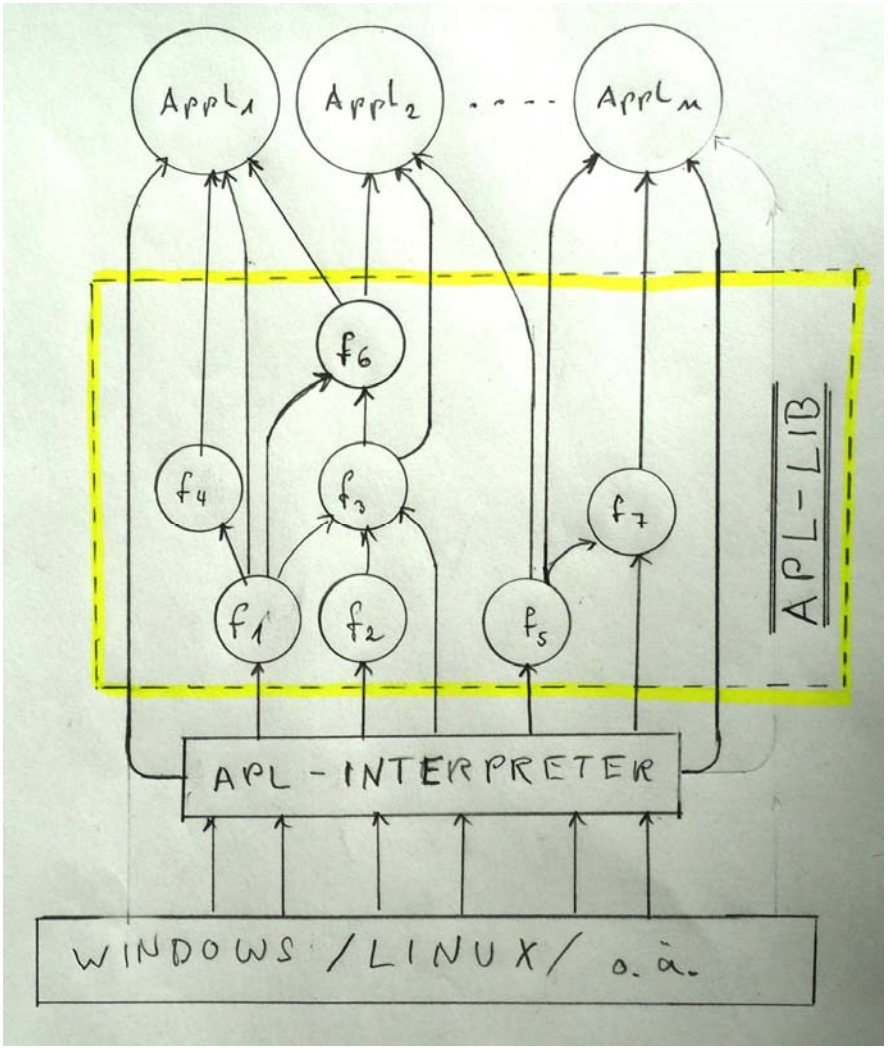
… oder wie man sich die Software-Entwicklung erleichtern kann

Ralf Herminghaus, Mai 2022

Warum eine APL-Library ?

1) In der Vergangenheit habe ich damit gute Erfahrungen gemacht
2) Das Thema wurde – aus welchen Gründen auch immer – in der APL-Community bisher vernachlässigt
3) Andere Programmiersprachen sind da deutlich weiter (C, FORTRAN, PYTHON,…)
   Oft taugt die Sprache selbst gar nicht viel, aber die vorhandenen Bibliotheken machen sie für viele Nutzer sehr nützlich.
4) In der Vergangenheit wurde in APL oft ein Programmierstil gepflegt, der es „uneingeweihten“ Lesern (d.h. Nicht-APL-Freaks) fast unmöglich machte, den Sinn eines APL-Programmes zu verstehen.

   Die Einführung von „allgemeinverständlichen“ Funktionen, deren



Zweck der Einführung einer APL-Programm-Library

- Besser lesbare Programme schreiben
- Annäherung an bereits bestehende Begriffe und Bezeichnungen
- Standard-Vorgänge zu Funktionen zusammenfassen
- Komplexität in den Applikationen reduzieren
- Portierungen vereinfachen

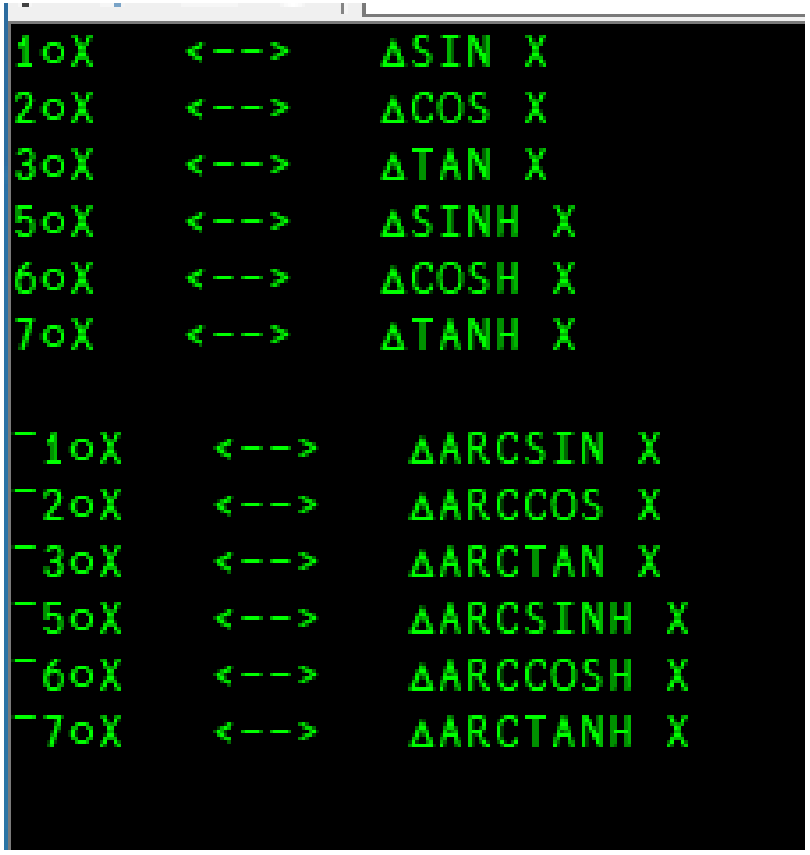
Begriffliche Vereinfachung durch Definition von Klassen und darauf definierten Operationen

Das dabei zugrundeliegende Paradigma ist die **funktionale Programmierung.**

Im Unterschied zur objektorientierten Programmierung können einmal erzeugte Objekte bei der funktionalen Programmierung ihren Zustand nicht mehr ändern.

Eine Funktion kann zwar ein Objekt X als Argument bekommen und ein ähnliches Objekt Y erzeugen. Den Zustand bzw. Wert von X kann die Funktion aber NICHT verändern.

Beispiel 1: Anpassung der APL-Bezeichungen an bestehende mathematische Standards



- Auch als APL-Enthusiast muss man zugeben, dass die rechte Formulierung für einen „Nicht-APL-er“ deutlich einfacher zu verstehen ist als die linke.

Beispiel2 : Vektor-wertiges IFF-Konstrukt zur besseren Lesbarkeit von Statements

- Für numerische Vektoren X1 und X2 findet man in APL-Programmen zuweilen Ausdrücke wie:

```
B←(X1>5)

a2←B×X1 + (1-B)×X2
```

- Besser verständlich wäre wahrscheinlich:

```
a2←ΔIFF (X1>5) X1 X2
```

Beispiel 3: Leichter lesbare Range-Funktionen

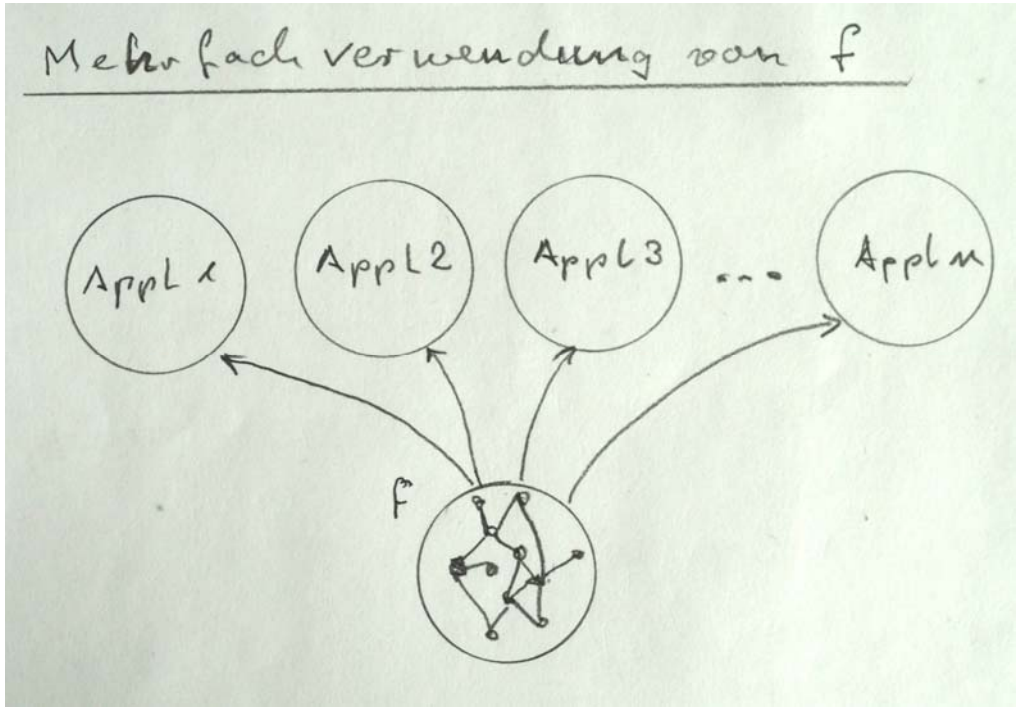
```
      3+((⍳(1+7-3))-⎕IO)
3 4 5 6 7
    (3 ΔTO 7)
3 4 5 6 7
```

```
      3+(⌽(⍳(1+7-3))-⎕IO)
7 6 5 4 3
    (7 ΔTO 3)
7 6 5 4 3
```

```
      1 ΔUP_TO 7
1 2 3 4 5 6 7
      7 ΔUP_TO 1
```

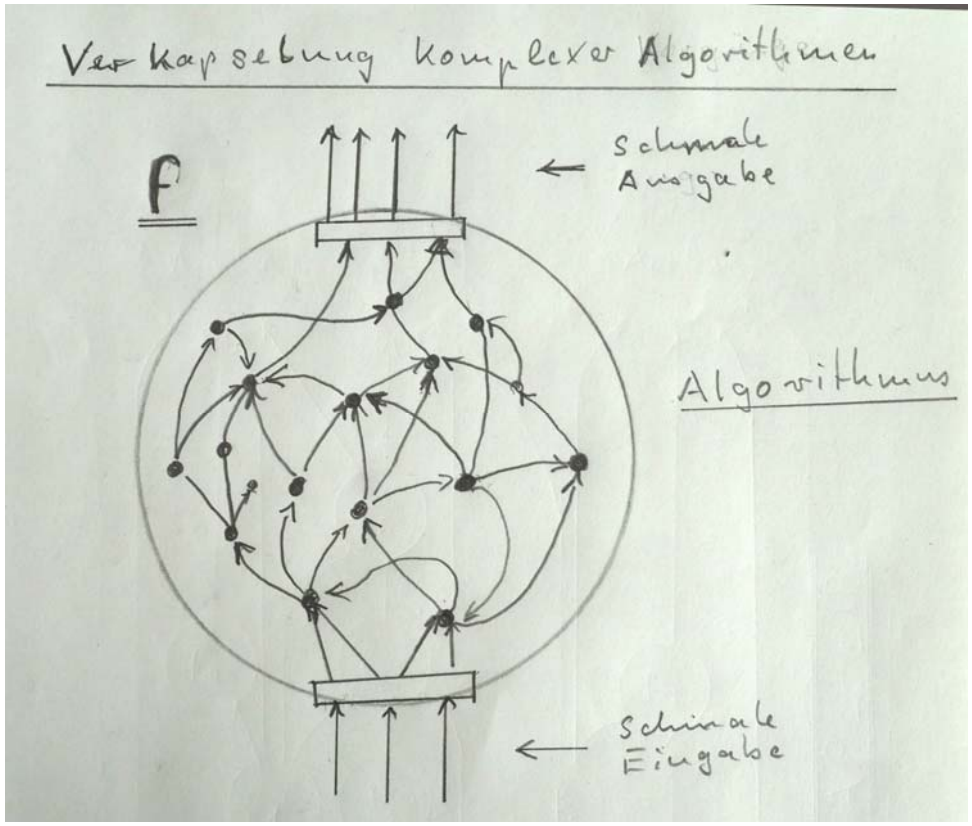
```
      1 ΔDOWN_TO 7


      7 ΔDOWN_TO 1
7 6 5 4 3 2 1
      ⌽(1 ΔDOWN_TO 7)
0
```

Beispiel 4: <u>Kapselung komplexer Vorgänge</u>



In Programmierkursen wird die Verwendung von Funktionen gerne damit motiviert, dass man einen gewissen Algorithmus nur einmal programmieren muss und ihn dann immer wieder verwenden kann.

Dies ist auch die wesentlichen Begründung der Benutzung von Programm-Librarys



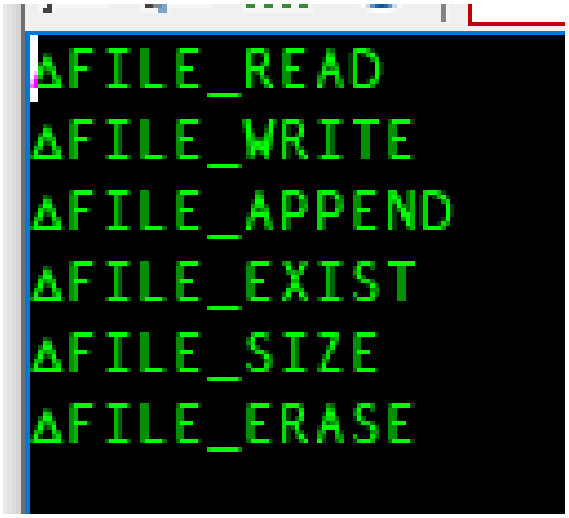
Ebenso wichtig ist aber auch der Aspekt, dass man komplexe Vorgänge, die den Leser des übergeordneten Programmes nicht interessieren, in einer „Kapsel“ versteckt, so dass der Text des übergeordneten Programmes einfacher lesbar wird.

Dieses Argument zieht auch dann, wenn die programmierte Funktion nur ein einziges Mal verwendet wird.

Konkretes Beispiel: Funktionen zum File-Handling

Lesen und Schreiben von Files geschieht immer in mehreren Einzelschritten die immer wieder der gleichen Logik folgen.
(z.B. File-Existenz prüfen, File binden, File lesen, File schließen ….).

Komplexe Arbeitsabläufe, die man gut zu handlichen Funktionen zusammenfassen kann:



```
[0]     LA ∆FILE_APPEND RA;⎕ML;⎕IO;⎕PP;⎕CT;FILE_NAME;FILE_INH;FILE_NR
[1]     A-------------------------------------------------------------
[2]     A Dipl.Math.Ralf Herminghaus, 27.1.2022
[3]     A    Append a Text-String to an existing File
[4]     A    Beachte: Die Funktion ist nur darauf ausgelegt,
[5]     A             einfache Strings an eine bestehende Datei anzuhängen
[6]     A
[7]     A Example:
[8]     A    a2←'''Twas brillig, and the slithy toves'
[9]     A    a2←a2,∆CRLF,'Did gyre and gimble in the wabe',∆CRLF
[10]    A
[11]    A    a3←'C:\APL_RH_LIB\Jabberwocky.txt'
[12]    A
[13]    A    ∆FILE_ERASE a3
[14]    A    a2 ∆FILE_APPEND a3
[15]    A    a2 ∆FILE_APPEND a3
[16]    A    a2 ∆FILE_APPEND a3
[17]    A
[18]    A-------------------------------------------------------------
[19]     ⎕ML←3 ⋄ ⎕IO←1 ⋄ ⎕PP←17 ⋄ ⎕CT←1E¯13
[20]    A-------------------------------------------------------------
[21]     FILE_NAME←RA
[22]     FILE_INH←LA
[23]
[24]     :If ⎕NEXISTS FILE_NAME
[25]         FILE_NR←FILE_NAME ⎕NTIE 0
[26]         FILE_INH ⎕NAPPEND FILE_NR
[27]         ⎕NUNTIE FILE_NR
[28]     :Else
[29]         FILE_NR←FILE_NAME ⎕NCREATE 0
[30]         FILE_INH ⎕NAPPEND FILE_NR
[31]         ⎕NUNTIE FILE_NR
[32]     :EndIf
```

In einzelnen Fällen ist es evtl. sinnvoll, allein aus Gründen der Kompatibilität eine eigene Funktion zu bilden:

```
[0]     Z←ΔFILE_EXIST RA;⎕ML;⎕IO;⎕PP;⎕CT;FILE_NAME
[1]     ⍝----------------------------------------------------
[2]     ⍝ Dipl.Math.Ralf Herminghaus, 20.4.2022
[3]     ⍝    Just for Compatibility-reasons .....
[4]     ⍝
[5]     ⍝ Example:
[6]     ⍝    a2←ΔFILE_EXIST 'C:\APL_RH_LIB\Jabberwocky.txt'
[7]     ⍝
[8]     ⍝----------------------------------------------------
[9]      ⎕ML←3 ⋄ ⎕IO←1 ⋄ ⎕PP←17 ⋄ ⎕CT←1E¯13
[10]    ⍝----------------------------------------------------
[11]     FILE_NAME←RA
[12]     Z←⎕NEXISTS FILE_NAME
```

Bei der Portierung der Funktion auf ein anderes APL-System könnte die Implementation dieser Funktion evtl. völlig anders ausehen.

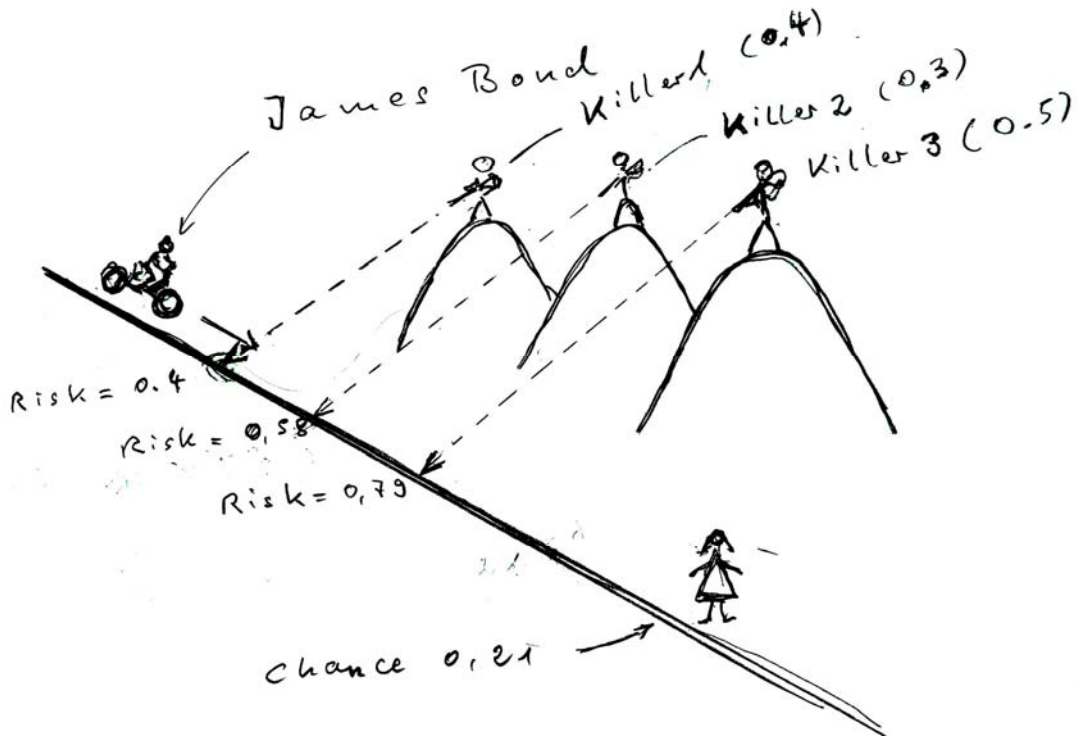
Konversion von **Text-Matrix** zu **Text-String** und umgekehrt

Texte in APL können in zwei Formen auftreten: **Matritzen** oder **Strings**.
Beide sind äquivalent und sollten leicht ineinander transformiert werden können

```
[0]     Z←ΔSTRING_FROM_MATRIX RA;TEXT_MATR;LINE_LENGHT;POS_BLANK
[1]   A-------------------------------------------------------
[2]   A Dipl.Math.Ralf Herminghaus, 28.10.2021
[3]   A    Change Text-Format Text-Matrix to Text-String
[4]   A
[5]   A Example:
[6]   A    a1←ΔSTRING_FROM_MATRIX (⎕CR 'ΔSTRING_FROM_MATRIX')
[7]   A
[8]   A-------------------------------------------------------
[9]     Z←TEXT_MATR←RA
[10]    →(1=ρρZ)/0
[11]    POS_BLANKS←' '=TEXT_MATR
[12]    LINE_LENGHT←+/~⍲∧\⍲POS_BLANKS
[13]    H1←⊂[2]TEXT_MATR
[14]    H2←LINE_LENGHT↑¨H1
[15]    Z←∊H2,¨(⊂⎕TC[2 3])
```

```
[0]     Z←ΔSTRING_TO_MATRIX RA;STRNG;SEL
[1]   A-------------------------------------------------------
[2]   A Dipl.Math.Ralf Herminghaus, 28.10.2021
[3]   A    Change Text-String to Text-Matrix
[4]   A
[5]   A Examples:
[6]   A    a1←'ABCDEFGHIJK',ΔCRLF,'LMNOPQ',ΔCRLF,'XXX'
[7]   A    a2←ΔSTRING_TO_MATRIX a1
[8]   A    ⍴a2
[9]   A
[10]  A    a1←'ABCDEFGHIJK',ΔCRLF,'LMNOPQ',ΔCRLF
[11]  A    a2←ΔSTRING_TO_MATRIX a1
[12]  A    ⍴a2
[13]  A
[14]  A-------------------------------------------------------
[15]    Z←STRNG←RA
[16]    →(2=ρρZ)/0
[17]    SEL←~STRNG∊(⎕TC[2 3])
[18]    Z←⊃SEL⊂STRNG
```

Beispiel 5: Eine spezielle Algebra zum Thema „Übersterblichkeit“

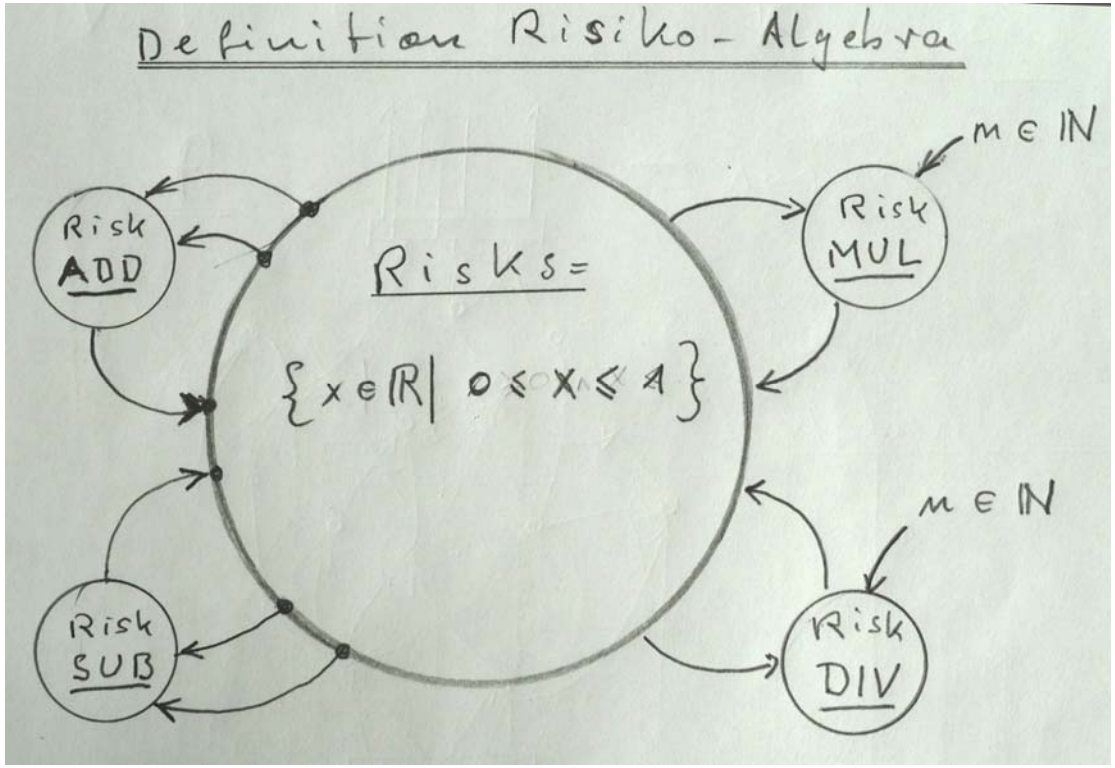


Die Addition von mehreren Einzelrisiken
ergibt zusammen immer noch ein Gesamt-Risiko von unter 1 .

Die dahinter liegende Logik kann man durch eine geeignete „Algebra“ sehr elegant darstellen:

```
(0.4 ΔRISK_ADD 0.3) ΔRISK_ADD 0.5
0.79
```

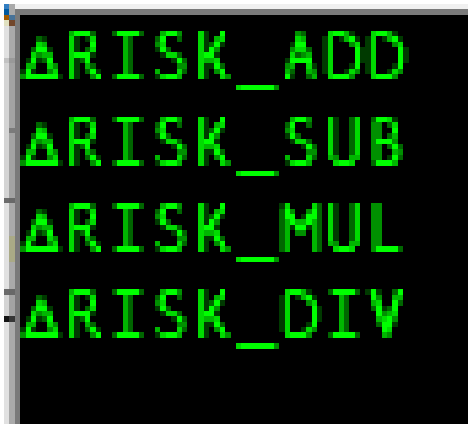
Chance = (1 – 0.4) x (1 – 0.3) x (1 – 0.5) = 0.21



Eine einfache Algebra zur Berechnung von Übersterblichkeits-Risiken läßt sich z.B. ohne großen Aufwand definieren. Die Risiken werden dabei als reelle Zahlen zwischen 0 und 1 modelliert.

Eine Definition für **Risiko-Addition** und **Risiko-Subtraktion** ergibt sich in natürlicher Weise.

Daraus abgeleitet ergeben sich dann auch sofort die Begriffe **Risiko-Multiplikation** und als Gegenstück die **Risiko-Division**.




```
[0]     Z←R1 ΔRISK_ADD R2
[1]   ⍝----------------------------------------
[2]   ⍝ Dipl.Math.Ralf Herminghaus, 7.7.2019
[3]   ⍝ Addition of Risks
[4]   ⍝
[5]   ⍝ Example:
[6]   ⍝   R1←0.4
[7]   ⍝   R2←0.3
[8]   ⍝   R3←R1 ΔRISK_ADD R2
[9]   ⍝
[10]  ⍝----------------------------------------
[11]    Z←1-(1-R1)×(1-R2)
```

```
[0]     Z←R1 ΔRISK_SUB R2
[1]   ⍝----------------------------------------
[2]   ⍝ Dipl.Math.Ralf Herminghaus, 7.7.2019
[3]   ⍝ Subtraction of Risks
[4]   ⍝
[5]   ⍝ Example:
[6]   ⍝   R1←0.4
[7]   ⍝   R2←0.3
[8]   ⍝   R3←R1 ΔRISK_ADD R2
[9]   ⍝   R4←R3 ΔRISK_SUB R2
[10]  ⍝   R1≡R4
[11]  ⍝
[12]  ⍝----------------------------------------
[13]    Z←1-(1-R1)÷(1-R2)
[14]
```

```
[0]     Z←R1 ΔRISK_MUL N
[1]   ⍝----------------------------------------
[2]   ⍝ Dipl.Math.Ralf Herminghaus, 7.7.2019
[3]   ⍝ Multiplikation of Risks
[4]   ⍝
[5]   ⍝ Example:
[6]   ⍝   R1←0.5
[7]   ⍝   R2←R1 ΔRISK_ADD R1
[8]   ⍝   R3←R1 ΔRISK_MUL 2
[9]   ⍝   R2≡R3
[10]  ⍝
[11]  ⍝   R4←R2 ΔRISK_ADD R1
[12]  ⍝   R5←R1 ΔRISK_MUL 3
[13]  ⍝   R4≡R5
[14]  ⍝
[15]  ⍝----------------------------------------
[16]    Z←1-(1-R1)*N
```

```
[0]     Z←R1 ΔRISK_DIV N
[1]   ⍝----------------------------------------
[2]   ⍝ Dipl.Math.Ralf Herminghaus, 7.7.2019
[3]   ⍝ Division of Risks
[4]   ⍝
[5]   ⍝ Example:
[6]   ⍝   R1←0.4
[7]   ⍝   R2←(R1 ΔRISK_ADD R1) ΔRISK_ADD R1
[8]   ⍝   R3←R1 ΔRISK_MUL 3
[9]   ⍝   R2≡R3
[10]  ⍝
[11]  ⍝   R4←R2 ΔRISK_DIV 3
[12]  ⍝   R4≡R1
[13]  ⍝
[14]  ⍝----------------------------------------
[15]    Z←1-(1-R1)*(1÷N)
```

Beispiel 6: Eine Algebra von Polynomen

Polynome der reellen Zahlen mit ganzzahligen Exponenten sind relativ gut bekannt. In mathematisch-technischen Anwendungen werden sie oft verwendet.

Mathematisch betrachtet bilden diese Polynome einen **Vektorraum**, so dass sich auf einem abstrakten Level ganz ähnliche Argumentationen wie in der Geometrie durchführen lassen.

Polynome eignen sich besonders gut als Objekte einer entsprechenden Algebra, auf der eine Menge von geeigneten Funktionen operieren kann.

Polynome einer Variablen X sind Funktionen der Form:

$P(X) = a_0 + a_{1x}X + a_{2x}(X*2) + a_{3x}(X*3) + \ldots.. + a_{nx}(X*n)$
$Q(X) = b_0 + b_{1x}X + b_{2x}(X*2) + b_{3x}(X*3) + \ldots.. + b_{nx}(X*n)$

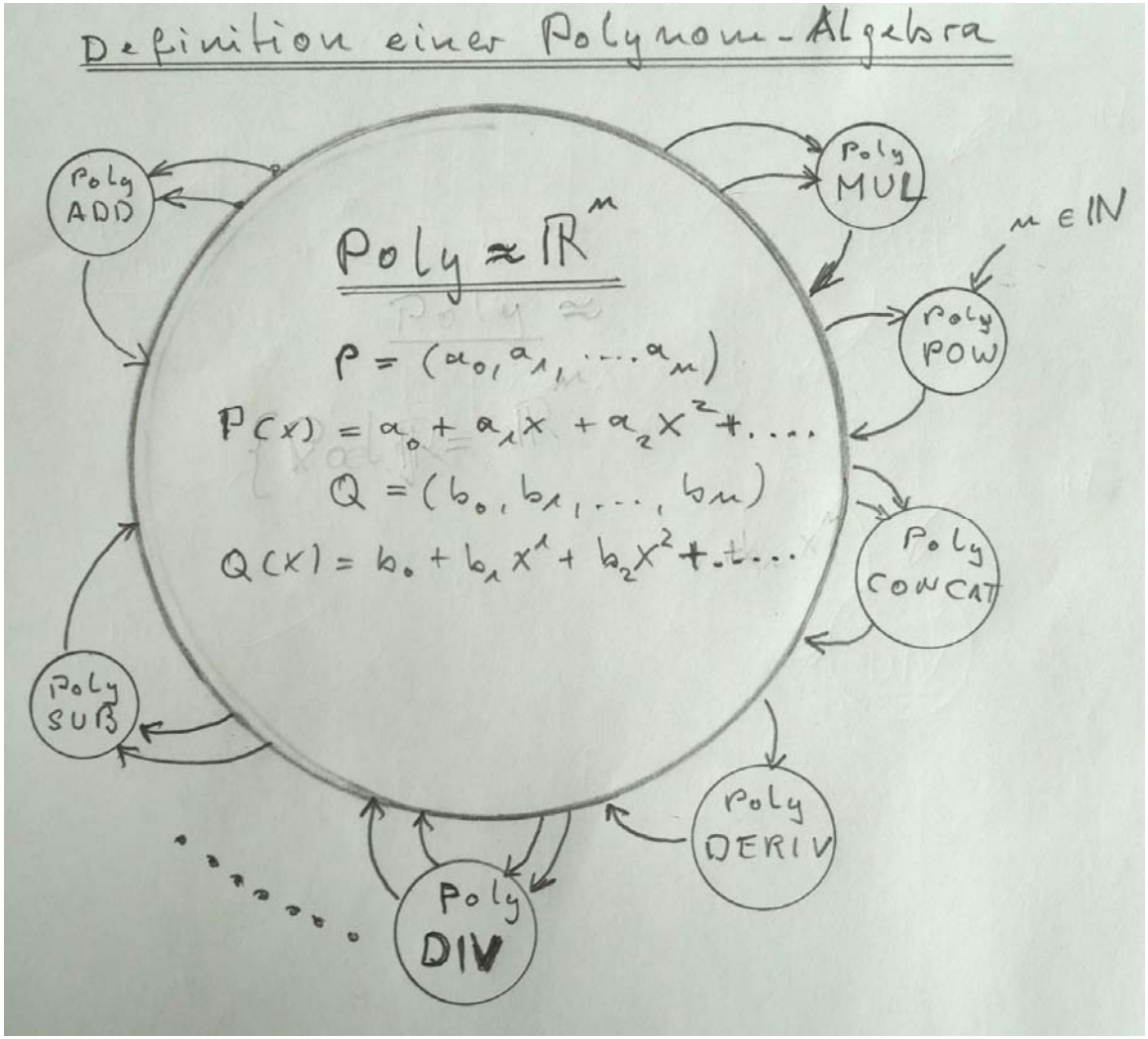
Bekanntermaßen gibt es eine triviale Addition und Subtraktion für Polynome:

$R = P + Q$ mit

$R(X) = (a_0+b_0) + (a_1+b_1)_xX + (a_2+b_2)_x(X*2) + (a_3+b_3)_x(X*3) + \ldots.. + (a_n+b_n)_x(X*n)$

$S = P - Q$ mit

$S(X) = (a_0-b_0) + (a_1-b_1)_xX + (a_2-b_2)_x(X*2) + (a_3-b_3)_x(X*3) + \ldots.. + (a_n-b_n)_x(X*n)$

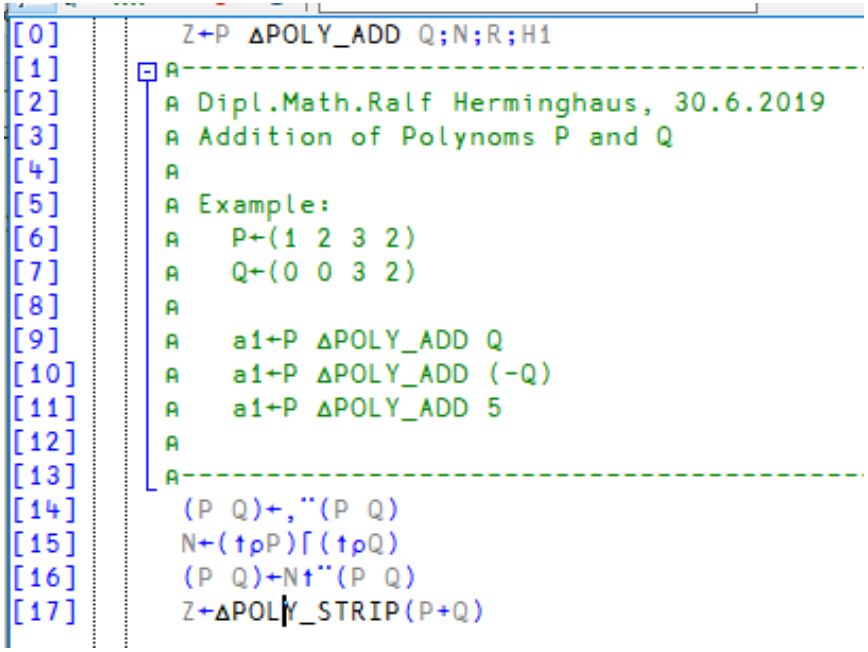


Was kann man mit Polynomen denn so alles machen ?

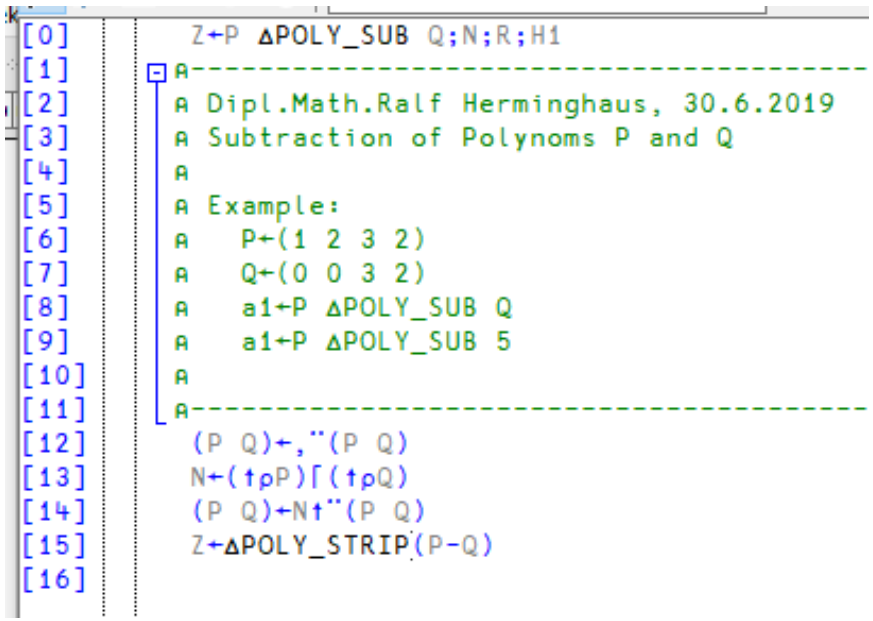
- Addieren
- Subtrahieren
- Multiplizieren
- Potenzen bilden
- Nacheinander ausführen

Aus der Schule kennt man vielleicht noch
- Ableiten
- Integrieren
- Polynomdivision

Modelliert man Polynome in **APL** als Vektoren von Zahlen,
ergibt sich die **Addition** und **Subtraktion** in natürlicher Weise:



```
[0]     Z←P ΔPOLY_ADD Q;N;R;H1
[1]   ⍝----------------------------------------
[2]   ⍝ Dipl.Math.Ralf Herminghaus, 30.6.2019
[3]   ⍝ Addition of Polynoms P and Q
[4]   ⍝
[5]   ⍝ Example:
[6]   ⍝    P←(1 2 3 2)
[7]   ⍝    Q←(0 0 3 2)
[8]   ⍝
[9]   ⍝    a1←P ΔPOLY_ADD Q
[10]  ⍝    a1←P ΔPOLY_ADD (-Q)
[11]  ⍝    a1←P ΔPOLY_ADD 5
[12]  ⍝
[13]  ⍝----------------------------------------
[14]    (P Q)←,¨(P Q)
[15]    N←(⍴P)⌈(⍴Q)
[16]    (P Q)←N↑¨(P Q)
[17]    Z←ΔPOLY_STRIP(P+Q)
```



```
[0]     Z←P ΔPOLY_SUB Q;N;R;H1
[1]   ⍝----------------------------------------
[2]   ⍝ Dipl.Math.Ralf Herminghaus, 30.6.2019
[3]   ⍝ Subtraction of Polynoms P and Q
[4]   ⍝
[5]   ⍝ Example:
[6]   ⍝    P←(1 2 3 2)
[7]   ⍝    Q←(0 0 3 2)
[8]   ⍝    a1←P ΔPOLY_SUB Q
[9]   ⍝    a1←P ΔPOLY_SUB 5
[10]  ⍝
[11]  ⍝----------------------------------------
[12]    (P Q)←,¨(P Q)
[13]    N←(⍴P)⌈(⍴Q)
[14]    (P Q)←N↑¨(P Q)
[15]    Z←ΔPOLY_STRIP(P-Q)
[16]
```

Was kann man denn noch sinnvolles mit Polynomen machen ?

Ganz wichtig: Ein Polynom für bestimmte Eingabe-Werte auswerten

```
[0]     Z←P ΔPOLY X;H1;H2;N;HP;I;Y
[1]     ⍝----------------------------------------
[2]     ⍝ Dipl.Math.Ralf Herminghaus, 30.6.2019
[3]     ⍝    Polynom-Calculation (Horner Algorithm)
[4]     ⍝
[5]     ⍝ Examples:
[6]     ⍝   P←(0 0 1 0 0)      ⍝   P(X)=(X×X)
[7]     ⍝   Q←(2 0 1 0 0)      ⍝   Q(X)=2 + (X×X)
[8]     ⍝   R←(0 0 0 1 0)      ⍝   R(X)=(X×X×X)
[9]     ⍝   S←(0 2 0 0 0)      ⍝   S(X)=(2×X)
[10]    ⍝
[11]    ⍝   a1←P ΔPOLY (⍳5)
[12]    ⍝   a2←Q ΔPOLY (⍳5)
[13]    ⍝   a3←R ΔPOLY (⍳5)
[14]    ⍝   a4←S ΔPOLY (⍳5)
[15]    ⍝
[16]    ⍝----------------------------------------
[17]      H1←~∧\(⌽P)=0
[18]      HP←H1/⌽P
[19]      N←⍴HP
[20]      Y←0
[21]      :For I :In (⍳N)
[22]          Y←Y+(I⊃HP)
[23]          :If I<N
[24]              Y←Y×X
[25]          :EndIf
[26]      :EndFor
[27]
[28]      Z←Y
```

Die „natürlichen“ Operationen, die man auf Polynomen ausführen kann
(vieles davon sollte man aus dem Mathematik-Unterricht der Oberstufe kennen)

```
ΔPOLY          A Auswertung
ΔPOLY_ADD      A Addition
ΔPOLY_SUB      A Subtraktion
ΔPOLY_MUL      A Multiplikation
ΔPOLY_DIV      A Polynomdivision mit Rest
ΔPOLY_POW      A Ganzzahlige Potenz R(X) = P(X) × P(X) × .... × P(X)
ΔPOLY_CONCAT   A Concatenation   R(X) =  P(Q(X))
ΔPOLY_DERIV    A Differential / Ableitung
ΔPOLY_INTEGR   A Integral
```

Viele differenzierbare Funktionen lassen sich durch Polynome gut approximieren:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \qquad \text{für alle } x$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + - \cdots \qquad -1 < x \leqq +1$$

$$\ln x = 2\left[\left(\frac{x-1}{x+1}\right) + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 + \cdots\right] \qquad x > 0$$

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + - \cdots \qquad \text{für alle } x$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + - \cdots \qquad \text{für alle } x$$

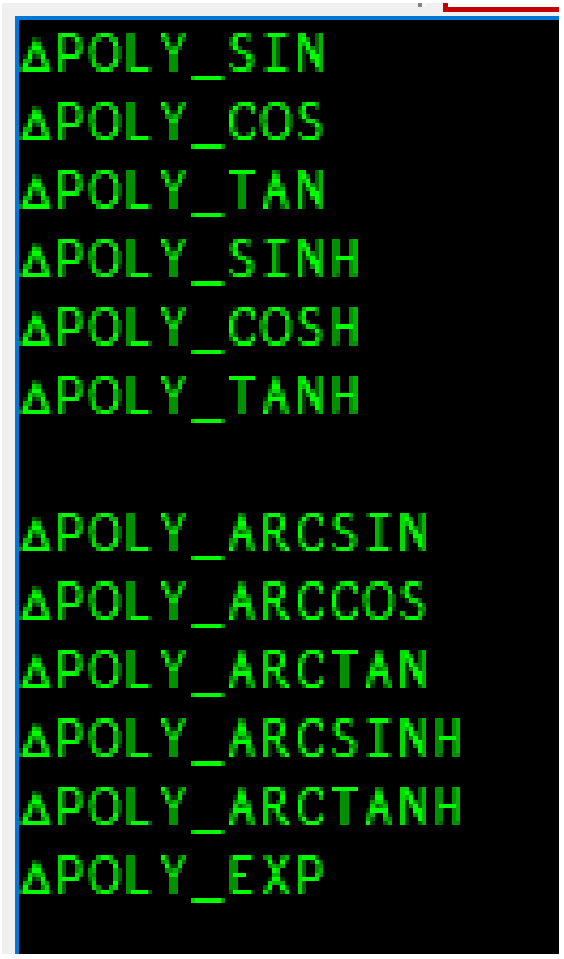
$$\arcsin x = x + \frac{1}{2}\cdot\frac{x^3}{3} + \frac{1\cdot 3}{2\cdot 4}\cdot\frac{x^5}{5} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot\frac{x^7}{7} + \cdots \qquad |x| \leqq 1$$

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + - \cdots \qquad |x| \leqq 1$$

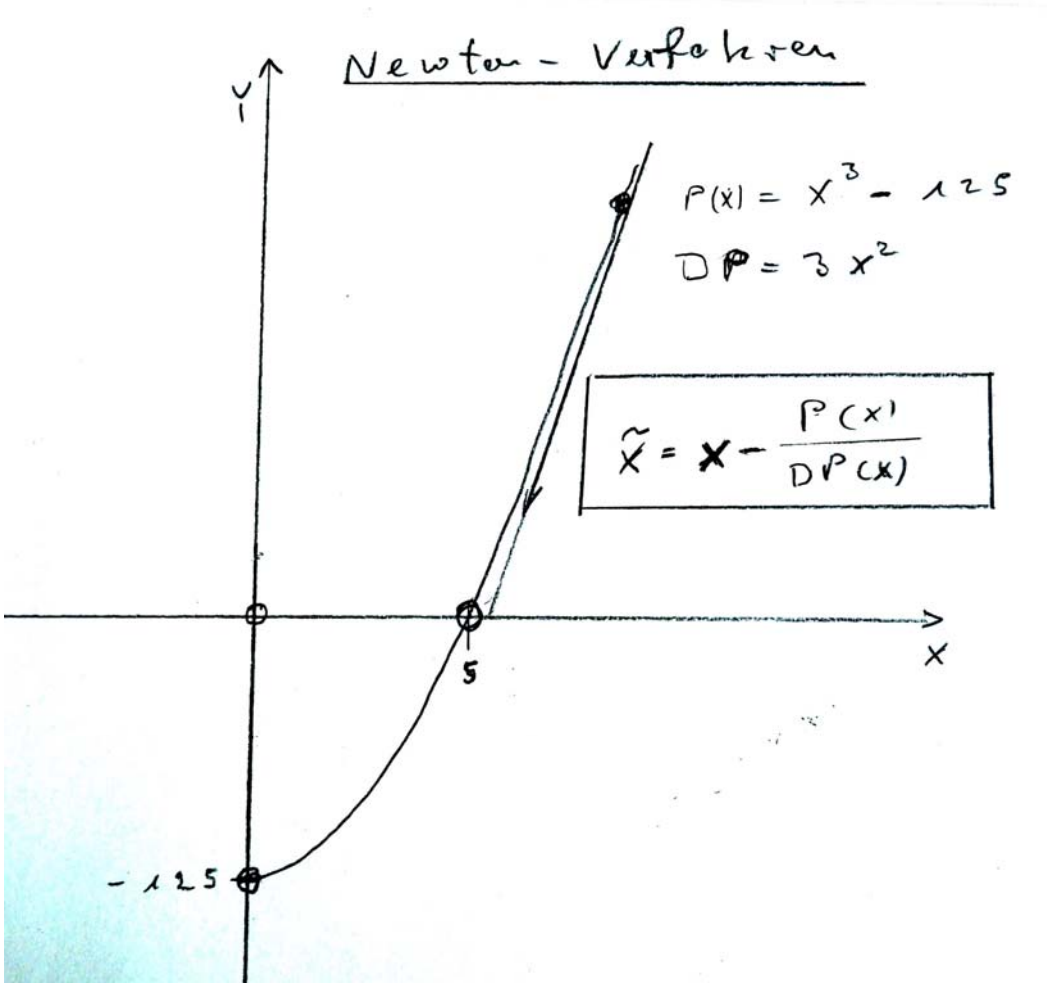
Entsprechend lassen sich Polynome definieren, die diese Funktionen näherungsweise ersetzen können.
Dabei gilt: Je länger das Polynom ist, desto akkurater ist die Approximationdesto höher aber auch der Rechenaufwand
In der APL-LIB kann man daher vorgeben, wie viele Terme das Näherungspolynom berücksichtigen soll. Hier gezeigt am Beispiel der SINUS-Funktion:

```
Dabei gibt das rechte Argument der Funktion an,
wieviele Koeffizienten für das Approximattions-
Polynom berücksichtigt werden sollen.

Beispielsweise beim SINUS:

∆POLY_SIN  3
0 1 0 ¯0.16666666666666666

     ∆POLY_SIN  5
0 1 0 ¯0.16666666666666666 0 0.008333333333333333

     ∆POLY_SIN  7
0 1 0 ¯0.16666666666666666 0 0.008333333333333333 0 ¯0.0001984126984126984

     ∆POLY_SIN  9
0 1 0 ¯0.16666666666666666 0 0.008333333333333333 0 ¯0.0001984126984126984 0 0.000002755731922398589
```

Beispiele für weitere Polynom-Approximationen:



Etwas komplexer: Nullstellensuche mit dem Newton-Verfahren



Eine angenehme Eigenschaft von Polynomen mit positiven ganzzahligen Exponenten ist u.a. die Tatsache, dass ihre Nullstellen vom Newton-Verfahren grundsätzlich immer gefunden werden.

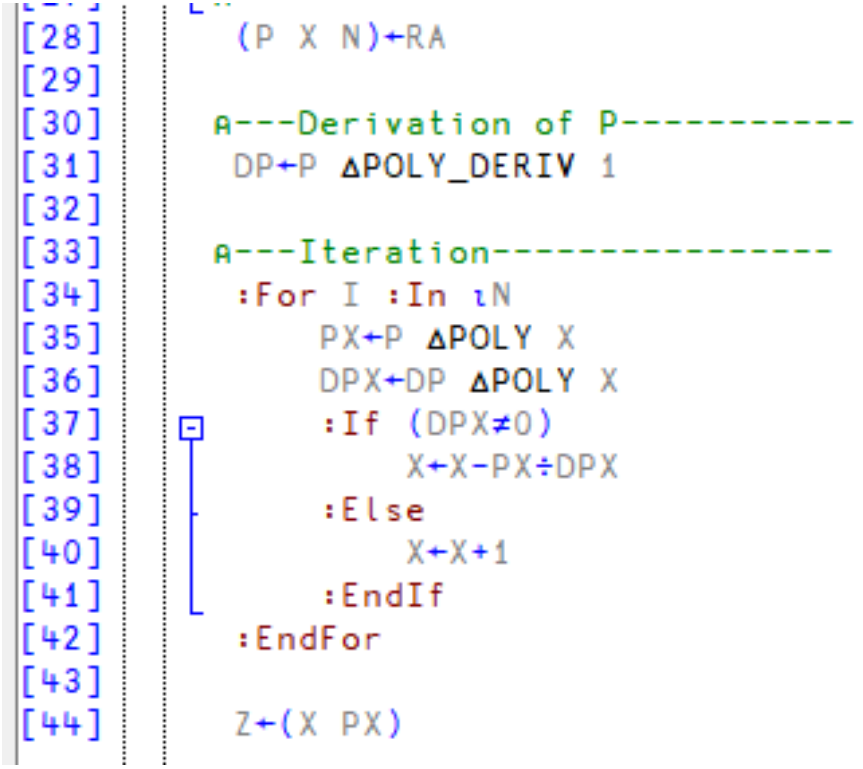
Gegenbeispiel dazu
wäre etwa die Funktion

F(x) = (Vorz(x)) * (Norm(x)**0.5)

deren Nullstelle vom Newton-Verfahren nicht gefunden wird.

```
[0]     Z←ΔPOLY_NEWTON RA;P;N;DP;I;X;PX;DPX
[1]     ⍝----------------------------------------
[2]     ⍝ Dipl.Math.Ralf Herminghaus, 1.3.2020
[3]     ⍝ Newton-Algorithm
[4]     ⍝
[5]     ⍝ P: Polynom
[6]     ⍝ X: Start-X
[7]     ⍝ N: Number of Iterations
[8]     ⍝
[9]     ⍝ Example1:
[10]    ⍝    P←(0 2 3 4 5 6)
[11]    ⍝    X←5
[12]    ⍝    N←20
[13]    ⍝    (X PX)←ΔPOLY_NEWTON (P X N)
[14]    ⍝
[15]    ⍝ Example2:  (Sqare-Root of 25)
[16]    ⍝    P←(¯25 0 1)
[17]    ⍝    X←1
[18]    ⍝    N←10
[19]    ⍝    (X PX)←ΔPOLY_NEWTON (P X N)
[20]    ⍝
[21]    ⍝ Example3:  (Third-Root of 125)
[22]    ⍝    P←(¯125 0 0 1)
[23]    ⍝    X←1
[24]    ⍝    N←10
[25]    ⍝    (X PX)←ΔPOLY_NEWTON (P X N)
[26]    ⍝
[27]    ⍝----------------------------------------
[28]      (P X N)←RA
[29]
```

```
[28]      (P X N)←RA
[29]
[30]    ⍝---Derivation of P----------
[31]      DP←P ΔPOLY_DERIV 1
[32]
[33]    ⍝---Iteration----------------
[34]      :For I :In ⍳N
[35]          PX←P ΔPOLY X
[36]          DPX←DP ΔPOLY X
[37]          :If (DPX≠0)
[38]              X←X-PX÷DPX
[39]          :Else
[40]              X←X+1
[41]          :EndIf
[42]      :EndFor
[43]
[44]      Z←(X PX)
```

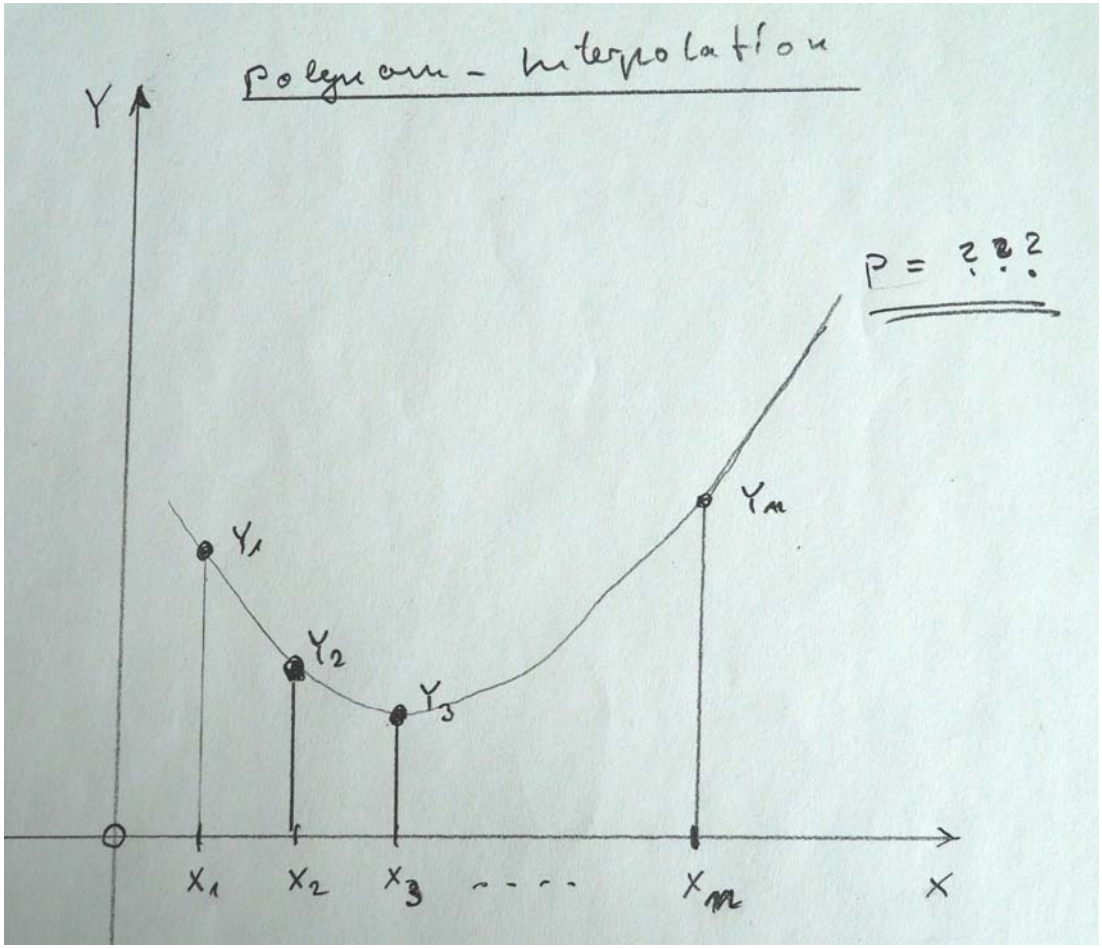


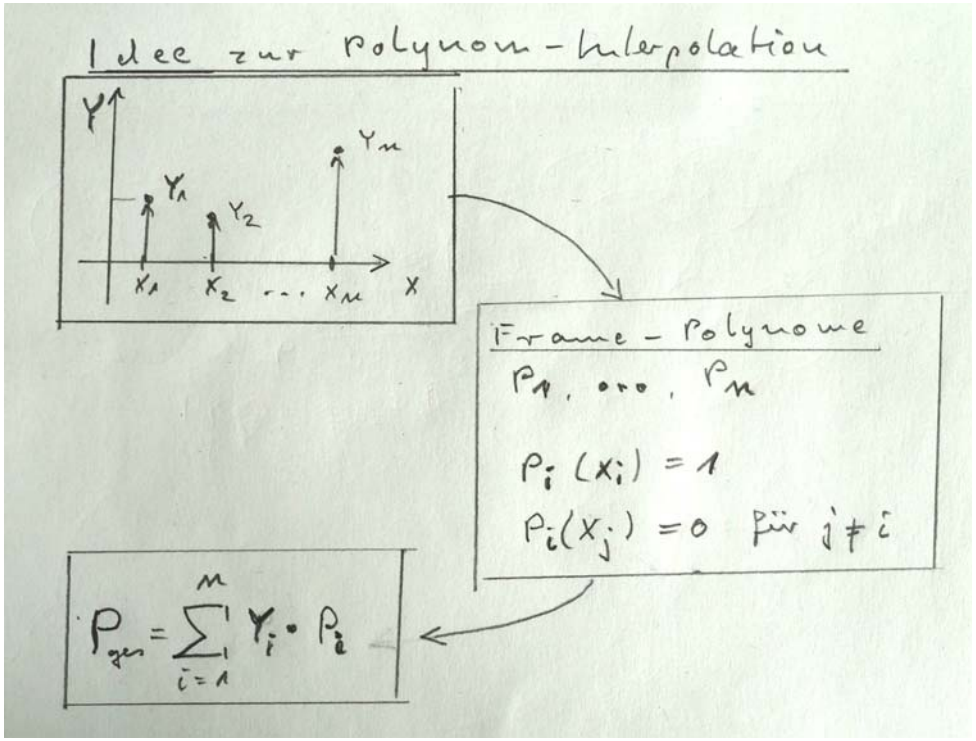
```
[28]     (P X N)←RA
[29]
[30]    ⍝---Derivation of P-----------
[31]     DP←P ΔPOLY_DERIV 1
[32]
[33]    ⍝---Iteration----------------
[34]     :For I :In ⍳N
[35]         PX←P ΔPOLY X
[36]         DPX←DP ΔPOLY X
[37]         :If (DPX≠0)
[38]             X←X-PX÷DPX
[39]         :Else
[40]             X←X+1
[41]         :EndIf
[42]     :EndFor
[43]
[44]     Z←(X PX)
```

<u>Beachte dabei:</u>

Das Newton-Verfahren verwendet bereits bestehende Funktionen **POLY_DERIV** und **POLY** für die Berechnung der ersten Ableitung und die Auswertung am Punkt X .

Noch etwas komplexer: Polynom-Interpolation

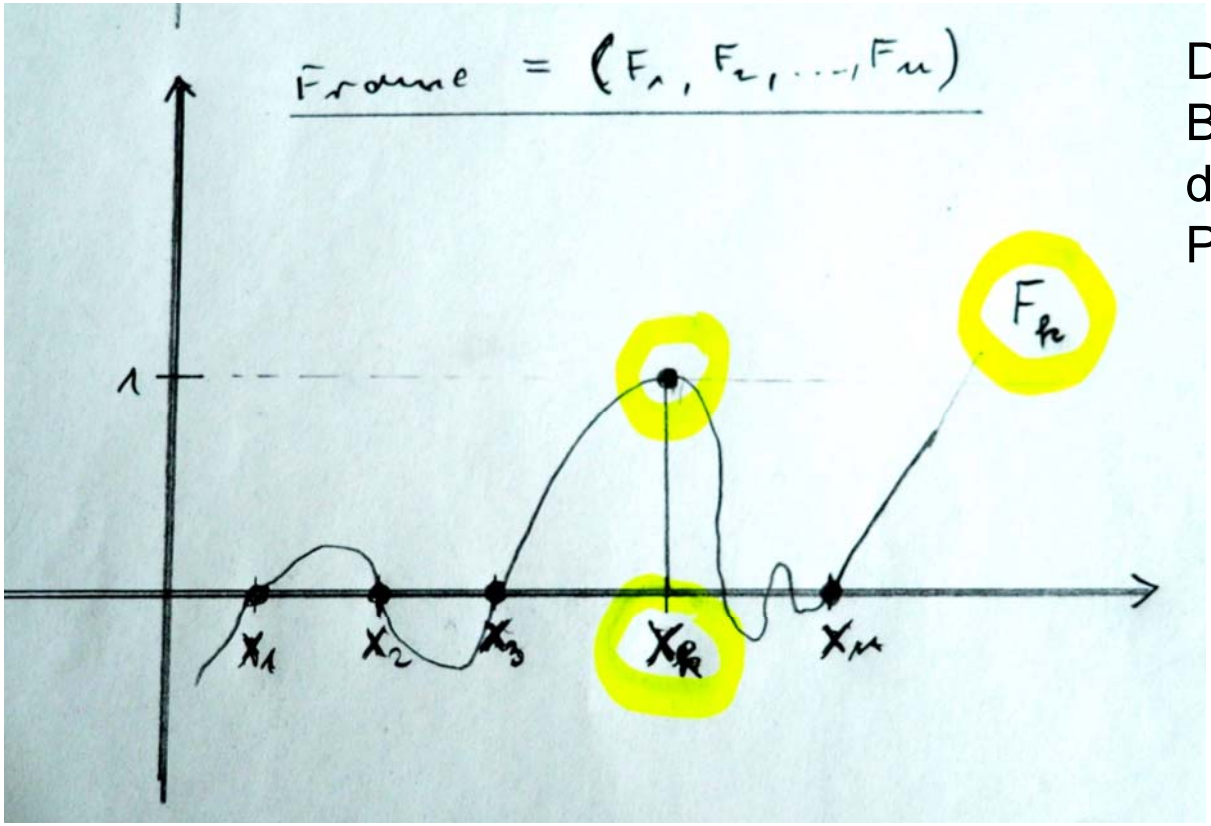




Die Grundidee bei der Polynom-Interpolation besteht darin, zunächst eine geeignete Menge (ein „Frame“) von Polynomen zu definieren, die im Vektorraum der Polynome eine n-dimensionale „Ebene“ aufspannt und bestimmte Eigenschaften besitzt (siehe links).

Die Interpolation selbst geschieht dann einfach durch Multiplikation mit Yi und Summation der dadurch entstehenden Polynome zu einem Ergebnis-Polynom

Jedes dieser Basis-Polynome hat nur für eine einzige Stützstelle den Wert 1,
bei allen anderen Stützstellen den Wert 0.



Durch Linearkombination dieser Basis-Polynome kann man nun leicht das gesuchte interpolierende Polynom konstruieren.

Der Algorithmus dazu:

```
[0]     Z←ΔPOLY_INTERPOL RA;⎕ML;⎕IO;⎕PP;⎕CT;X;Y;FRAME;
[1]   ⍝----------------------------------------------
[2]   ⍝ Dipl.Math.Ralf Herminghaus, 29.2.2020
[3]   ⍝ Polynomial Interpolation
[4]   ⍝
[5]   ⍝ Example1: (Approximation of X*2)
[6]   ⍝    X←⍳10
[7]   ⍝    Y←X*2
[8]   ⍝    P←ΔPOLY_INTERPOL X Y
[9]   ⍝    PX←P ΔPOLY X
[10]  ⍝    a1←Y-PX
[11]  ⍝
[12]  ⍝ Example2: (Approximation of ΔSIN)
[13]  ⍝    X←(0 10 ΔETC_TO 180)
[14]  ⍝    Y←(360 ΔSIN X)
[15]  ⍝    P←ΔPOLY_INTERPOL X Y
[16]  ⍝    PX←P ΔPOLY X
[17]  ⍝    a1←Y-PX
[18]  ⍝
[19]  ⍝ Example3: (Approximation of ΔSQRT)
[20]  ⍝    X←⍳10
[21]  ⍝    Y←ΔSQRT X
[22]  ⍝    P←ΔPOLY_INTERPOL X Y
[23]  ⍝    PX←P ΔPOLY X
[24]  ⍝    a1←Y-PX
[25]  ⍝
[26]  ⍝ Example4:  (Approximation of ΔARCSIN)
[27]  ⍝    X←(0 10 ΔETC_TO 90)
[28]  ⍝    Y←(360 ΔSIN X)
[29]  ⍝    P←ΔPOLY_INTERPOL Y X
[30]  ⍝    PY←P ΔPOLY Y
[31]  ⍝    a1←PY-(360 ΔARCSIN Y)
```

```
[33]    ⍝----------------------------------
[34]      ⎕ML←3 ⋄ ⎕IO←1 ⋄ ⎕PP←17 ⋄ ⎕CT←1E¯13
[35]    ⍝----------------------------------
[36]      (X Y)←RA
[37]
[38]      FRAME←ΔPOLY_FRAME X
[39]      Z←P_INTERPOL←∈+/FRAME×Y
```

Die wesentliche Arbeit steckt dabei natürlich in der Bestimmung der Basis-Polynome für den Frame … :-)

Bestimmung der notwendigen Basis-Polynome

```
[0]   Z←ΔPOLY_FRAME RA;⎕ML;⎕IO;⎕PP;⎕CT;X_LIST;N;I;P_GES;PI;FRA
[1]   A-------------------------------------------------------
[2]   A Dipl.Math.Ralf Herminghaus, 22.4.2022
[3]   A    Standard-Basis im Raum der Polynome
[4]   A    Diese Basis dient der Interpolation mit Polynomen
[5]   A    Vorgegeben wird eine Liste mit X-Werten
[6]   A    Heraus kommt eine Liste von Polynomen, die jeweils
[7]   A    beim Wert XI den Wert 1 annehmen, in allen anderen
[8]   A    Werten der Liste aber Null sind.
[9]   A
[10]  A Example:
[11]  A    a1←⍳5
[12]  A    a2←ΔPOLY_FRAME a1
[13]  A
[14]  A    a3←(1⊃a2) ΔPOLY a1
[15]  A    a3←(2⊃a2) ΔPOLY a1
[16]  A    a3←(3⊃a2) ΔPOLY a1
[17]  A    a3←(4⊃a2) ΔPOLY a1
[18]  A    a3←(5⊃a2) ΔPOLY a1
[19]  A
[20]  A-------------------------------------------------------
[21]    ⎕ML←3 ⋄ ⎕IO←1 ⋄ ⎕PP←17 ⋄ ⎕CT←1E¯13
```

```
[22]  A-------------------------------------------------------
[23]   X_LIST←RA
[24]
[25]   N←⍴X_LIST
[26]   P1←(1 0)
[27]   FRAME←N⍴⊂P1
[28]
[29]   :For I :In ⍳N
[30]       PI←((-I⊃X_LIST)1)
[31]       :For J :In ⍳N
[32]           :If J≠I
[33]               (J⊃FRAME)←(J⊃FRAME)ΔPOLY_MUL PI
[34]           :EndIf
[35]       :EndFor
[36]   :EndFor
[37]
[38]   A---Normierung auf 1-----------------------
[39]   :For I :In ⍳N
[40]       PI_XI←(I⊃FRAME)ΔPOLY(I⊃X_LIST)
[41]       :If PI_XI≠0
[42]           (I⊃FRAME)←(I⊃FRAME)÷PI_XI
[43]       :EndIf
[44]   :EndFor
[45]
[46]   Z←FRAME
```

An diesem Algorithmus sieht man beispielhaft recht schön, wie sich komplexe Sachverhalte durch Verwendung geeigneter Unterfunktionen knapp und übersichtlich formulieren lassen.

Ziel dabei muss immer sein, einem zukünftigen Leser des Programmes das Verständnis zu erleichtern – soweit der Sachverhalt es zuläßt.

Die Reihe der Beispiele ließe sich so endlos fortführen -
Leider reicht die Zeit dafür aber nicht aus :-(.

Zudem steht die Arbeit an dieser APL-Programm-Bibliohek ja erst am Anfang…..

Weitere Themen, die in der Planung sind:

- Polynom-Approximation
- Funktionen zum Thema **Quaternionen**
- Funktionen zum Thema **Oktonionen**
- **Tensoren** (z.B. Tucker-Zerlegung)
- Grafische Darstellung **4-dimensionaler Objekte**
- Funktionen zur **Workspace-Verwaltung**
- Funktionen zum Thema **CSV-Daten-Tabellen**
- und, und, und …….

Ideen und Anregungen sind immer herzlich willkommen!!!

…… Und vielleicht kommt ja irgendwann auch mal
was brauchbares dabei heraus …..

--- THE END ---