

Einführung in die Computer-Algebra

Teil 2 : Fortgeschrittenere Möglichkeiten mit *Mathematica*

Informatik I – Mathematik mit dem Computer
Prof. Dr. Alfred Schreiber
Institut für Mathematik und ihre Didaktik · Universität Flensburg

Numerisches Lösen von Gleichungen

```
Remove["Global`*"]
```

■ Vorbereitung

Linke Seite einer Gleichung zweckmäßigerweise als Funktion definieren:

```
g[x_] := x^3 - 2 x + 1
```

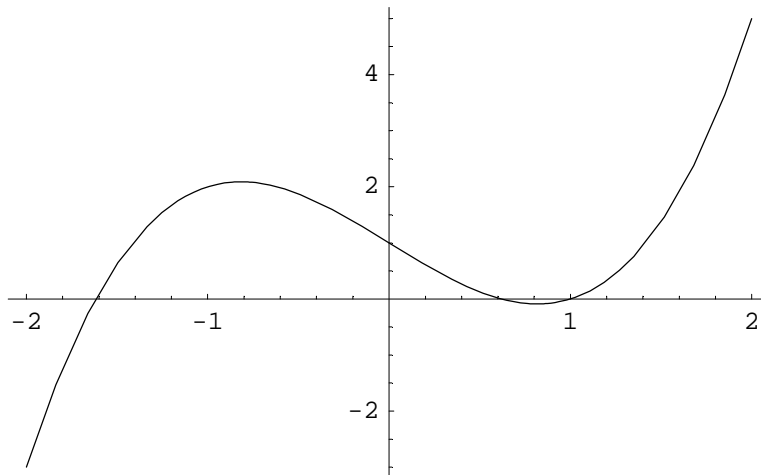
Gleichung durch Symbol (glg) wiedergeben:

```
glg = g[x] == 0
```

```
1 - 2 x + x^3 == 0
```

■ Reeller Funktionsgraph

```
Plot[g[x], {x, -2, 2}];
```



Grafik markieren (Mausklick) und anschließend mittels [Strg]-Taste + Mauscursor die Lage der Nullstellen angenähert messen.

In diesem Fall lassen sich die 3 reellen Nullstellen von $g(x)$ auch algebraisch berechnen:

```
x /. Solve[glg, x]
```

```
{1,  $\frac{1}{2}(-1 - \sqrt{5})$ ,  $\frac{1}{2}(-1 + \sqrt{5})$ }
```

■ Numerische Lösungen

Erste Möglichkeit (nur für Polynome!):

```
x /. NSolve[glg, x]
```

```
{-1.61803, 0.618034, 1.}
```

Die zweite (allgemeinere) Möglichkeit verlangt Angabe eines Startwerts (für das eingebaute Newtonsche Verfahren):

```
FindRoot[glg, {x, 0}]
FindRoot[glg, {x, 2}]
FindRoot[glg, {x, -1}, WorkingPrecision -> 30, AccuracyGoal -> 30]
```

```
{x -> 0.618033}
```

```
{x -> 1.}
```

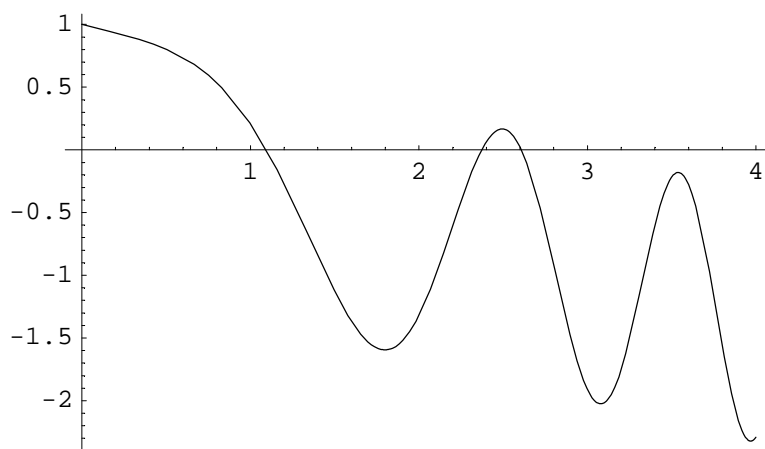
```
{x -> -1.61803398874989484820458683437}
```

■ Beispiel: Eine transzendente Gleichung

```
f[x_] := Cos[x2] -  $\frac{x}{3}$ ;
```

Schaubild zur Lokalisierung reeller Nullstellen:

```
Plot[f[x], {x, 0, 4}];
```



```
glg = f[x] == 0;
```

Die Startwerte können nun so gewählt werden, dass *Mathematica* die im Schaubild gesichteten reellen Lösungen findet:

```
FindRoot[glg, {x, 1}]
```

```
{x -> 1.09427}
```

```
FindRoot[glg, {x, 2}]
```

```
{x -> 2.3715}
```

Durch den Startwert 3 wird keine neue Nullstelle gefunden:

```
FindRoot[glg, {x, 3}]
```

```
{x -> 2.3715}
```

Man braucht einen Startwert zwischen den Nullstellen:

```
FindRoot[glg, {x, 2.4, 3}]
```

```
{x -> 2.60774}
```

Operationen mit Listen

```
Remove["Global`*"]
```

■ Erweiterte Zugriffsverfahren

Eine Beispiel-Liste

(inhaltlich sinnlos, aber zweckmäßig für die folgende Demonstration):

```
mliste = {0, -1, "dummy", Sqrt[5], Pi, 2/3, -1, {1, 2, 3}, {}}
```

```
{0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}}
```

Die ersten 3 Elemente:

```
Take[mliste, 3]
```

```
{0, -1, dummy}
```

Die letzten 3 Elemente:

```
Take[mliste, -3]
```

```
{-1, {1, 2, 3}, {}}
```

Erstes und letztes Element:

```
First[mliste]  
Last[mliste]
```

```
0
```

```
{}
```

Elemente nach "Eigenschaften" auswählen (reine Funktionen mit Wahrheitswert; siehe weiter unten).

$\sqrt{5}$ und π sind (ohne numerische Auswertung) Symbole und werden nicht als Zahlen erkannt:

```
Select[mliste, NumberQ]
```

```
{0, -1,  $\frac{2}{3}$ , -1}
```

```
Select[mliste, IntegerQ]
```

```
{0, -1, -1}
```

```
Select[mliste, ListQ]
```

```
{{1, 2, 3}, {}}
```

■ Zugehörigkeit und Position

Ist -1 Element der Liste mliste?

```
MemberQ[mliste, -1]
```

```
True
```

An welcher Position kommt das Element -1 vor?

Mathematica gibt eine Liste aus, die die Positionen in Listenform enthält:

```
Position[mliste, -1]
```

```
{{2}, {7}}
```

■ Anhängen eines Elements

Element a anhängen:

```
AppendTo[mliste, a]
```

```
{0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

mliste enthält das neue Element a:

```
mliste
```

```
{0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

Der folgende Befehl lässt mliste unverändert:

```
mliste2 = Append[mliste, 704]
```

```
{0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a, 704}
```

```
mliste
```

```
{0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

■ Element am Listenanfang einfügen

```
PrependTo[mliste, 13]
```

```
{13, 0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

Alternativ (ohne Änderungen von mlist):

```
Prepend[mliste, b]
```

```
{b, 13, 0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

```
mliste
```

```
{13, 0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

■ Einfügen und Löschen

Element 1000 an Position 7 einfügen:

```
Insert[mliste, 1000, 7]
```

```
{13, 0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ , 1000,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

Das eingefügte Element befindet sich **nur in der von Insert ausgegebenen Liste**, aber **nicht** in der als Parameter übergebenen Liste mliste:

```
mliste
```

```
{13, 0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

Das Element an Position 5 von mliste löschen (und das Ergebnis dieser Operation in neuelliste speichern):

```
neuelliste = Delete[mliste, 5]
```

```
{13, 0, -1, dummy,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

Zum Vergleich die alte und die neue Liste:

```
mliste  
neuelliste
```

```
{13, 0, -1, dummy,  $\sqrt{5}$ ,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

```
{13, 0, -1, dummy,  $\pi$ ,  $\frac{2}{3}$ , -1, {1, 2, 3}, {}, a}
```

■ Listen als Mengen

Tabula rasa ...

```
Remove["Global`*"]
```

Eine Liste mit mehrfachen Element-Vorkommen:

```
aliste = {5, 2, 4, 6, 5, 2, 3, 4, 4, 1};
```

Die Menge der Elemente von aliste:

```
amenge = Union[aliste]
```

```
{1, 2, 3, 4, 5, 6}
```

■ Einebnen einer Liste

Eine Liste, der Elemente z.T. Listen sind, usw.

```
mliste = {1, {1, 4}, {3, {1, 2}}};
```

Einebnen heißt: Die Objekte auf der untersten Ebene sammeln

```
Flatten[mliste]
```

```
{1, 1, 4, 3, 1, 2}
```

■ Vereinigung

```
amenge
```

```
{1, 2, 3, 4, 5, 6}
```

```
bmenge = Table[k, {k, 4, 10}]
```

```
{4, 5, 6, 7, 8, 9, 10}
```

Vereinigung als Liste!

```
cmenge = Join[amenge, bmenge]
```

```
{1, 2, 3, 4, 5, 6, 4, 5, 6, 7, 8, 9, 10}
```

■ Durchschnitt

```
Intersection[amenge, bmenge]
```

```
{4, 5, 6}
```

```
Intersection[bmenge, cmenge]
```

```
{4, 5, 6, 7, 8, 9, 10}
```

■ Komplement

```
Complement[bmenge, amenge]
```

```
{7, 8, 9, 10}
```

■ Kartesisches Produkt

Zwei Ausgangslisten:

```
aliste = {a, b, c};  
bliste = {1, 2, 3, 4, 5};
```

```
akreuzb = Outer[List, aliste, bliste]
```

```
{{{a, 1}, {a, 2}, {a, 3}, {a, 4}, {a, 5}},  
 {{b, 1}, {b, 2}, {b, 3}, {b, 4}, {b, 5}},  
 {{c, 1}, {c, 2}, {c, 3}, {c, 4}, {c, 5}}}
```

Achtung: Es handelt sich nicht um die Menge aller Paare!

```
Length[akreuzb]
```

```
3
```

Erst die folgende Operation (Einebnen in der 1. Stufe) liefert das übliche kartesische Mengenprodukt:

```
Flatten[akreuzb, 1]
```

```
{{a, 1}, {a, 2}, {a, 3}, {a, 4}, {a, 5}, {b, 1}, {b, 2},
 {b, 3}, {b, 4}, {b, 5}, {c, 1}, {c, 2}, {c, 3}, {c, 4}, {c, 5}}
```

Schleifen

```
Remove["Global`*"]
```

■ Do-Schleife

Wiederholung einer Operation mittels **Do** (nach dem Vorbild von Table, Sum etc.):

```
Do[a[i] = i^2, {i, 1, 10, 2}]
```

Das Ergebnis sichtbar machen (der Befehl **Array[a, n]** erzeugt eine Liste von n Elementen der Form **a[i]**):

```
Array[a, 10]
```

```
{1, a[2], 9, a[4], 25, a[6], 49, a[8], 81, a[10]}
```

■ For-Schleife

Ein (aus der herkömmlichen Programmierung bekannter) Schleifentyp, bei dem die Anzahl der Schleifendurchläufe (wie bei der Do-Schleife) vorgegeben ist.

Bem.: Die For-Schleife ist entbehrlich.

```
For[i = 1, i ≤ 5, i++, Print[a[i]]]
```

```
1  
a[2]  
9  
a[4]  
25
```

■ While-Schleife

`While[bedingung, aktion]` bedeutet:

Solange *bedingung* wahr ist, führe *aktion* aus.

```
i = 1;  
While[Mod[i, 5] ≠ 0, (i = i + 3; Print[i])];
```

```
4  
7  
10
```

N.B.: Wenn bei Eintritt in die Schleife die Bedingung nicht wahr ist, wird die Schleife nicht durchlaufen (sog. abweisende Schleife).

Man setze etwa im obigen Beispiel $i = 5$.

Definition von Funktionen

```
Remove["Global`*"]
```

■ Reine Funktionen

Eine Funktion wie $f(x) = 5x^2 - x + 1$ lässt sich wie folgt definieren:

```
f[x_] := 5 x^2 - x + 1
```

```
f[a]
```

```
1 - a + 5 a^2
```

Eine alternative und häufig benutzte Definitionsform ist die einer sog. reinen Funktion.

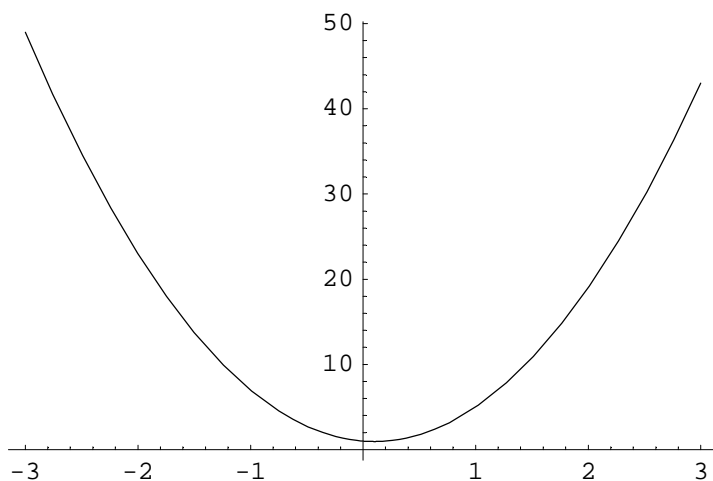
Idee dazu: f = die Funktion, welche x den Wert $5x^2 - x + 1$ zuordnet

```
g = Function[x, 5 x^2 - x + 1]
```

```
Function[x, 5 x^2 - x + 1]
```

g kann in der üblichen Weise benutzt werden:

```
Plot[g[u], {u, -3, 3}];
```



Eine andere Form der Definition reiner Funktionen

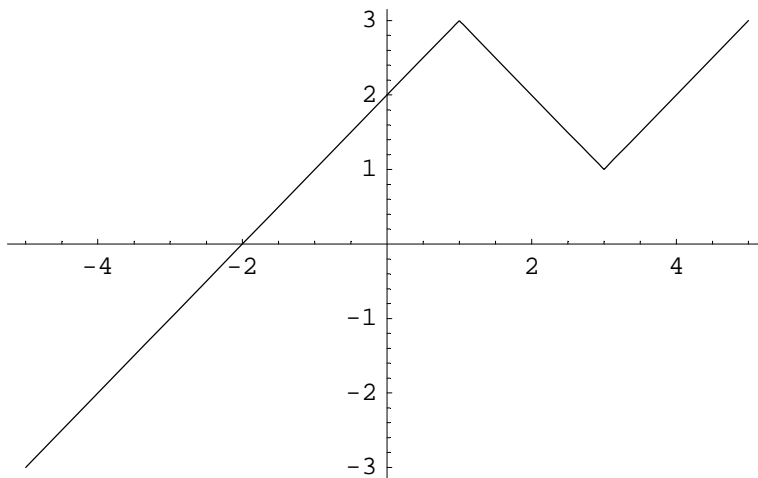
(# ist der Platzhalter für das Argument, & markiert das Ende der Definition):

```
h = # + Abs[3 - #] - Abs[1 - #] &
```

```
#1 + Abs[3 - #1] - Abs[1 - #1] &
```

Abs[.] ist der Absolutbetrag.

```
Plot[h[x], {x, -5, 5}];
```



■ Kompilierte Funktionen

Kompilation bedeutet: Übersetzung in maschinennahen Code. Sie dient der Beschleunigung von Berechnungen.

Kompilierte Funktionen werden wie reine Funktionen definiert (mittels Compile statt Function):

```
fc = Compile[x, x^3 Cos[x]];
f = Function[x, x^3 Cos[x]];
```

Vergleich der Rechenzeit:

```
f[4.8] // Timing
fc[4.8] // Timing
```

```
{0. Second, 9.67669}
```

```
{0. Second, 9.67669}
```

```
Clear[f, fc, g]
```

Unterschiede zeigen sich erst, wenn zahlreiche Operationen wiederholt ausgeführt werden sollen:

Beispiel einer unkomplierten Funktion:

```
g[m_] := Sum[Sum[Mod[i, k], {i, 1, m}], {k, 1, m}]
```

```
g[1000] // Timing
```

```
{15.82 Second, 225771449}
```

Das zugehörige Kompilat erweist sich als deutlich schneller:

```
gc = Compile[n, Sum[Sum[Mod[i, k], {i, 1, n}], {k, 1, n}]];
```

```
gc[1000] // Timing
```

```
{1.15 Second, 225771449}
```

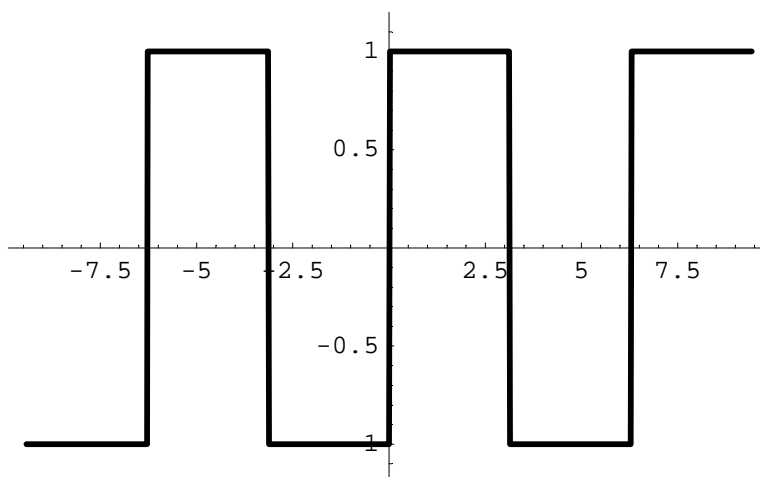
■ Bedingte Definitionen

Fallunterscheidung mittels `If[_,_,_]`

(geeignet nur für 2 Fälle):

```
f[x_] := If[Sin[x] > 0, 1, -1]
```

```
Plot[f[x], {x, -3 Pi, 3 Pi}, PlotRange -> {-1.2, 1.2}, PlotStyle -> {Thickness[0.008]}];
```



Fallunterscheidung mittels `Which`

(Vorteil: mehr als 2 Fälle möglich):

```
vorz[x_] := Which[
  x < 0, -1,
  x == 0, 0,
  x > 0, 1];
```

```
vorz[-7]
```

```
-1
```

Eine flexible und bequeme Form der bedingten Definition durch den Operator /;

```
Clear[f];
f[n_] := 0 /; n < 0;
f[n_] := 1 /; Not[IntegerQ[n]];
f[n_] := 2 /; EvenQ[n];
f[n_] := Sqrt[n]
```

```
{f[-3], f[Sqrt[5]], f[14], f[7]}
```

```
{0, 1, 2,  $\sqrt{7}$ }
```

■ Rekursive Definitionen

Rekursive Definition der Fibonacci-Folge:

```
Clear[f]
f[n_] := f[n - 1] + f[n - 2]
f[0] = f[1] = 1;
```

```
Table[f[i], {i, 10}]
```

```
{1, 2, 3, 5, 8, 13, 21, 34, 55, 89}
```

```
f[25] // Timing
```

```
{3.63 Second, 121393}
```

Neudefinition als Funktion "mit Gedächtnis":

```
Clear[f]
f[n_] := f[n] = f[n - 1] + f[n - 2]
f[0] = f[1] = 1;
```

```
f[25] // Timing
```

```
{0. Second, 121393}
```

```
$RecursionLimit = 2000;
```

```
f[1000] // Timing
```

```
(* Berechnung erfordert Löschen von f und Neudefinition *)
```

```
{0.11 Second,
 70330367711422815821835254877183549770181269836358732742604905087154533
 711819693357974224949456261173348775044924176599108818636326545022364
 710601205337412127386733911119813937312559876769009190224524532340350
 1}
```

■ Module

Bei vielen Berechnungen genügt es nicht, einen Formelausdruck auszuwerten; oft sind mehrere (von Bedingungen abhängige) Rechenschritte nacheinander auszuführen.

In *Mathematica* lassen sich dynamische Berechnungen dieser Art durch das **Module**[.]-Konstrukt wiedergeben.

Beispiel: Zentralwert einer Stichprobe

```
s = {2, 4, 4, 3, 1, 5, 4, 3, 2, 5, 6, 2, 3};
```

Die Liste wird nach Rangordnung (hier: Größe) sortiert:

```
sord = Sort[s]
```

```
{1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 6}
```

Länge ermitteln:

```
slen = Length[s]
```

```
13
```

Element "in der Mitte" (Position 7) suchen (Zentralwert oder Median):

```
Part[sord, Quotient[slen, 2] + 1]
```

```
3
```

Diese Rechenschritte sollen a) zusammengefasst und b) allgemein dargestellt werden:

```
Zentralwert[s_] := Module[{sord, slen, mpos, zwert},
  sord = Sort[s];
  slen = Length[s];
  mpos = Quotient[slen, 2];
  zwert = If[OddQ[slen], sord[[mpos + 1]], (sord[[mpos]] + sord[[mpos + 1]]) / 2];
  zwert
]
```

Die Probe auf's Exempel:

```
Zentralwert[s]
```

```
3
```

Eine Stichprobe von gerader Anzahl:

```
Zentralwert[{2, 1, 6, 1, 3, 3}]
```

```
 $\frac{5}{2}$ 
```

Wichtige Bemerkung:

Innerhalb von Module erklärte Symbole (Variable) haben nur lokale Gültigkeit (d.h. sind außerhalb des Moduls nicht bekannt).

Packages

```
Remove["Global`*"]
```

■ Vorgefertigte Pakete nutzen

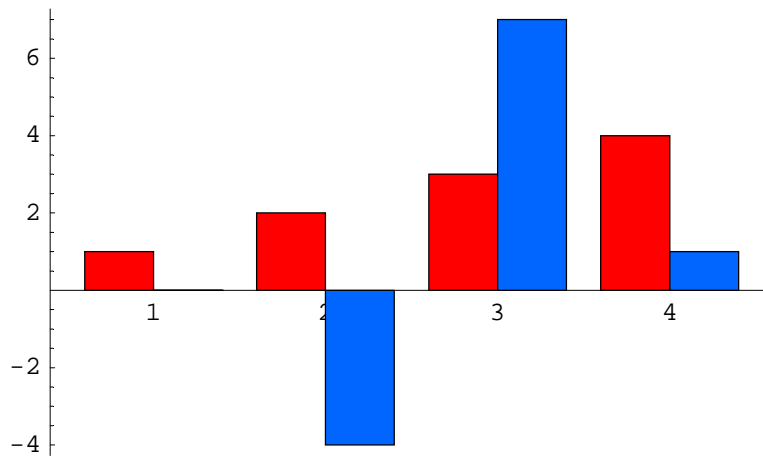
Nicht alle Funktionen von *Mathematica* sind bereits nach dem Start des Kerns verfügbar. Sie lassen sich aber durch Laden sog. Packages nachträglich verfügbar machen.

Ein Beispiel aus dem **Standard-Paket** *Graphics*

Lade-Befehl (**Needs**["...`...`"] oder kürzer <<...`...`)

```
Needs["Graphics`Graphics`"]
```

```
BarChart[{1, 2, 3, 4}, {0, -4, 7, 1}];
```



■ Eigene Pakete entwickeln

Ein Package ist eine spezielle Datei mit der Endung `.m`

Sie kann automatisch erzeugt werden, wenn man die Befehle (Funktionen), die das Paket enthalten soll, in Initialisierungszellen schreibt.

Als Demonstrationsbeispiel diene ein Mini-Package mit Funktionen zur Mengenalgebra. Es knüpft an den Abschnitt "Operationen mit Listen" an.

Mengenoperator:

```
(* Init *)
Menge[a_List] := Union[a]
```

```
Menge[{9, 9, 4, 4, 3, 2, 0}]
```

```
{0, 2, 3, 4, 9}
```

Ist das Argument keine Liste, bewirkt Menge nichts:

```
Menge[2]
```

```
Menge[2]
```

Vereinigungsmenge:

```
(* Init *)
Vereinigungsmenge[a_List, b_List] := Menge[Join[a, b]]
```

```
Vereinigungsmenge[{1, 4, 2, 5}, {3, 4, 5, 6, 1}]
```

```
{1, 2, 3, 4, 5, 6}
```

Schnittmenge:

```
(* Init *)
Schnittmenge[a_List, b_List] := Menge[Intersection[a, b]]
```

```
Schnittmenge[{1, 4, 2, 5}, {3, 4, 5, 6, 1}]
```

```
{1, 4, 5}
```

Differenzmenge:

```
(* Init *)
Differenzmenge[a_List, b_List] := Menge[Complement[a, b]]
```

Produktmenge:

```
(* Init *)
Produktmenge[a_List, b_List] := Flatten[Outer[List, Menge[a], Menge[b]], 1]
```

```
Produktmenge[{1, 2}, {a, b, c}]
```

```
{{1, a}, {1, b}, {1, c}, {2, a}, {2, b}, {2, c}}
```

Die mit (* Init *) gekennzeichneten Zellen können nun in eine neue Notebook-Datei **Mengenalgebra.nb** kopiert und dort als Initialisierungszellen ausgezeichnet werden. Beim Abspeichern kann man entscheiden, ob das zugehörige Paket **Mengenalgebra.m** automatisch erzeugt werden soll.

■ Dokumentation, Bereitstellung

Selbstentwickelte Packages sollten dokumentiert werden. Sie werden dann durch bestimmte technische Maßnahmen bereitgestellt (vgl. dazu die *Mathematica*-Hilfe sowie die angegebene Literatur [R. Maeder](#) sowie [M.-L. Herrmann](#)).