Mathematik mit dem Computer

■ *Mathematica*−Texte und mathematische Texte

Zum Austausch mit anderen Dokumenten ist es oftmals nützlich, die Ausgaben der Kommandos nicht in der zweidimensionalen **OutputForm**, sondern in der eindimensionalen **InputForm** ausgeben zu lassen.

```
$PrePrint = InputForm;
Expand[(x + y)^3]
```

```
x^3 + 3*x^2*y + 3*x*y^2 + y^3
```

```
Clear[$PrePrint]
```

```
Expand[(x + y)^3]
```

$$x^3 + 3\,x^2\,y + 3\,x\,y^2 + y^3$$

■ Mathematische Experimente und mathematische Beweise

Mathematik zu treiben bedeutet, einerseits mathematische *Regelmäßigkeiten und Zusammenhänge* zu entdecken und andererseits diese Beobachtungen mit (streng deduktiven Ansprüchen genügenden) *Beweisen* zu Gesetzmäßigkeiten zu verdichten. Für beide Phasen des "Doing Mathematics" bietet *Mathematica* vielfältige Unterstützung.

■ Beispiel 1: Potenzreste

```
3^100
```

```
515 377 520 732 011 331 036 461 129 765 621 272 702 107 522 001
```

```
3^10^10
```

```
General::ovfl : Overflow occurred in computation.
```

```
Overflow[]
```

```
Table[Mod[3^k, 10], {k, 0, 40}]
```

```
{1, 3, 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1, 3,
 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1}
```

```
Table[Mod[3^k, 7], {k, 0, 40}]
```

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

```
Table[Mod[3^k, 17], {k, 0, 40}]
```

```
{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1, 3, 9, 10,
 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1, 3, 9, 10, 13, 5, 15, 11, 16}
```

```
PowerMod[3, 10^10, 10]
```

```
1
```

```
PowerMod[3, 10^10, 10^15]
```

```
552 200 000 000 001
```

■ Beispiel 2: Perfekte Zahlen

```
DivisorSum[n_Integer] := Apply[Plus, Divisors[n]]

For[i = 2, i < 800, i++,
 If[DivisorSum[i] == 2 i, Print[i]]]
```

```
6

28

496
```

```
Select[Range[800], (DivisorSum[#] == 2 #) &]
```

```
{6, 28, 496}
```

```
tb = Table[With[{n = 2^(k-1) (2^k - 1)},
    If[DivisorSum[n] == 2 n, {k, n, "ja"}, {k, n, "nein"}]], {k, 2, 40}];
```

```
Grid[Prepend[tb, {"k", "n", "Perfekte Zahl"}],
 Dividers → {{{True}}, {True, True, {False}, True}},
 Spacings → {4, {2, 2, {.5}, 2}},
 Alignment → {{{Right}, Center}}]
```

k	n	Perfekte Zahl
2	6	ja
3	28	ja
4	120	nein
5	496	ja
6	2016	nein
7	8128	ja
8	32 640	nein
9	130 816	nein
10	523 776	nein
11	2 096 128	nein
12	8 386 560	nein
13	33 550 336	ja
14	134 209 536	nein
15	536 854 528	nein
16	2 147 450 880	nein
17	8 589 869 056	ja
18	34 359 607 296	nein
19	137 438 691 328	ja
20	549 755 289 600	nein
21	2 199 022 206 976	nein
22	8 796 090 925 056	nein
23	35 184 367 894 528	nein
24	140 737 479 966 720	nein
25	562 949 936 644 096	nein
26	2 251 799 780 130 816	nein
27	9 007 199 187 632 128	nein
28	36 028 796 884 746 240	nein
29	144 115 187 807 420 416	nein
30	576 460 751 766 552 576	nein
31	2 305 843 008 139 952 128	ja
32	9 223 372 034 707 292 160	nein
33	36 893 488 143 124 135 936	nein
34	147 573 952 581 086 478 336	nein
35	590 295 810 341 525 782 528	nein
36	2 361 183 241 400 462 868 480	nein
37	9 444 732 965 670 570 950 656	nein
38	37 778 931 862 819 722 756 096	nein
39	151 115 727 451 553 768 931 328	nein
40	604 462 909 806 764 831 539 200	nein

■ Beispiel 3: Der Vietasche Wurzelsatz

```
s = NSolve[x^7 - 3 x^6 + x - 1 == 0, x]
```

```
{{x → -0.756147 - 0.459404 i}, {x → -0.756147 + 0.459404 i},
 {x → 0.0660789 - 0.861295 i}, {x → 0.0660789 + 0.861295 i},
 {x → 0.691445 - 0.305089 i}, {x → 0.691445 + 0.305089 i}, {x → 2.99725}}
```

```
Apply[Plus, x /. s]
```

```
3. + 0. i
```

```
s = x /. Solve[x^2 + 2 a x + b == 0, x]
```

$$\left\{-a-\sqrt{a^2-b}\,,\ -a+\sqrt{a^2-b}\right\}$$

```
s[[1]] + s[[2]]
```

```
-2 a
```

```
s[[1]] s[[2]] // Expand
```

```
b
```

```
Table[Collect[Product[x - x_i, {i, 1, k}], x], {k, 2, 5}]
```

$$\left\{x^2+x\,(-x_1-x_2)+x_1\,x_2,\ x^3+x^2\,(-x_1-x_2-x_3)-x_1\,x_2\,x_3+x\,(x_1\,x_2+x_1\,x_3+x_2\,x_3),\right.$$
$$x^4+x^3\,(-x_1-x_2-x_3-x_4)+x_1\,x_2\,x_3\,x_4+x^2\,(x_1\,x_2+x_1\,x_3+x_2\,x_3+x_1\,x_4+x_2\,x_4+x_3\,x_4)+$$
$$x\,(-x_1\,x_2\,x_3-x_1\,x_2\,x_4-x_1\,x_3\,x_4-x_2\,x_3\,x_4),\ x^5+x^4\,(-x_1-x_2-x_3-x_4-x_5)-$$
$$x_1\,x_2\,x_3\,x_4\,x_5+x^3\,(x_1\,x_2+x_1\,x_3+x_2\,x_3+x_1\,x_4+x_2\,x_4+x_3\,x_4+x_1\,x_5+x_2\,x_5+x_3\,x_5+x_4\,x_5)+$$
$$x^2\,(-x_1\,x_2\,x_3-x_1\,x_2\,x_4-x_1\,x_3\,x_4-x_2\,x_3\,x_4-x_1\,x_2\,x_5-x_1\,x_3\,x_5-x_2\,x_3\,x_5-x_1\,x_4\,x_5-$$
$$\left.x_2\,x_4\,x_5-x_3\,x_4\,x_5)+x\,(x_1\,x_2\,x_3\,x_4+x_1\,x_2\,x_3\,x_5+x_1\,x_2\,x_4\,x_5+x_1\,x_3\,x_4\,x_5+x_2\,x_3\,x_4\,x_5)\right\}$$

■ Programmierung

Mathematica stellt eine voll ausgebaute imperative Programmiersprache zur Verfügung, die um (effizient implementierte) Elemente einer funktionalen Programmiersprache erweitert ist, und die vor allem beim Arbeiten mit rekursiv definierten Datenstrukturen wie z.B. Listen hilfreich sind. Der eingebaute Patternmatcher erlaubt regelbasiertes Programmieren, das besonders für die gezielte Simplifikation von Ausdrücken eingesetzt werden kann. Konzepte der Modularisierung, Kapselung, Paketbildung sowie inkrementelle Kompilierung stehen nicht nur den Entwicklern, sondern auch den Anwendern zur Verfügung.

■ Steuerstrukturen

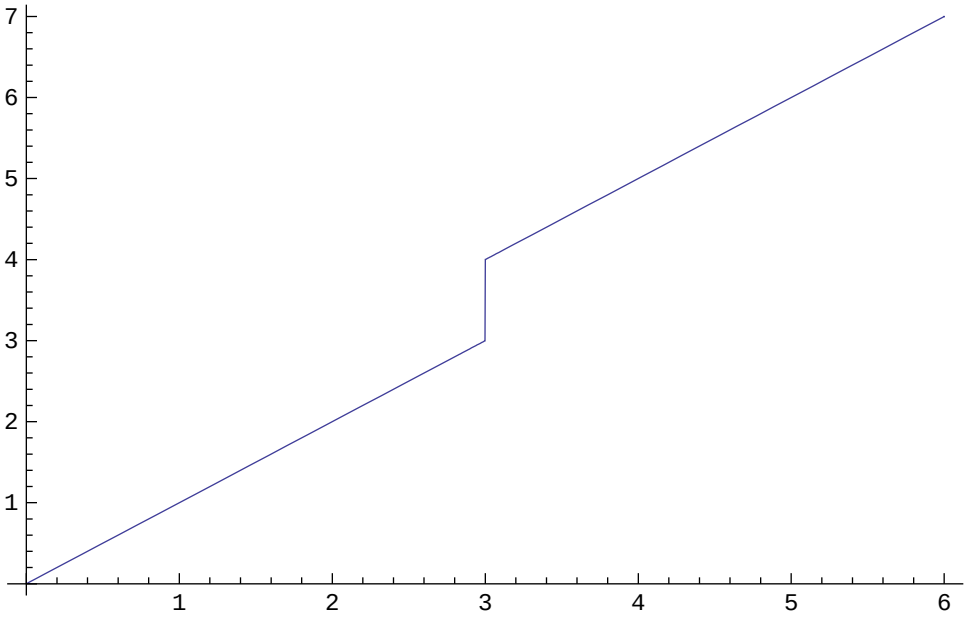
```
f[x_] := If[x < 3, x, x + 1]
```

```
g[x_] := Piecewise[{{x, x < 3}}, x + 1]
```

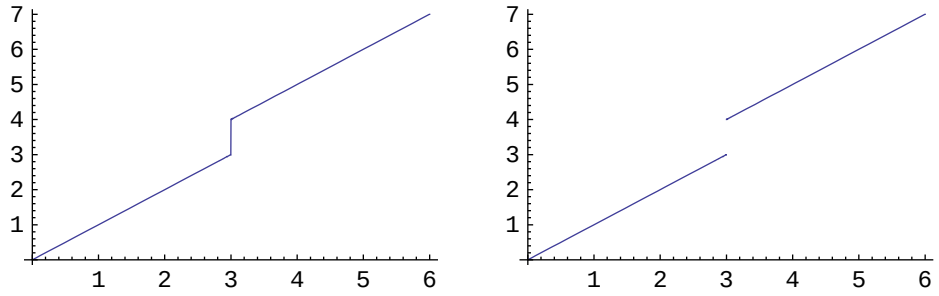
oder

```
g[x_] := Piecewise[{{x, x < 3}}, x + 1]
```

```
Plot[f[x], {x, 0, 6}]
```




```
GraphicsGrid[{{Plot[f[x], {x, 0, 6}], Plot[g[x], {x, 0, 6}]}}]
```



Meist werden die mathematischen EIgenschaften stückweiser Funktionen von **Piecewise** präziser berücksichtigt als in **If**−Konstrukten.
Hier etwa wird die Ableitung mathematisch präziser ermittelt und auch deren Graph ist an den Sprungstellen richtig dargestellt.

```
f1[x_] := If[x < 1, 2 x + 3, 6 - x^2]
f2[x_] := Piecewise[{{2 x + 3, x < 1}}, 6 - x^2]
```

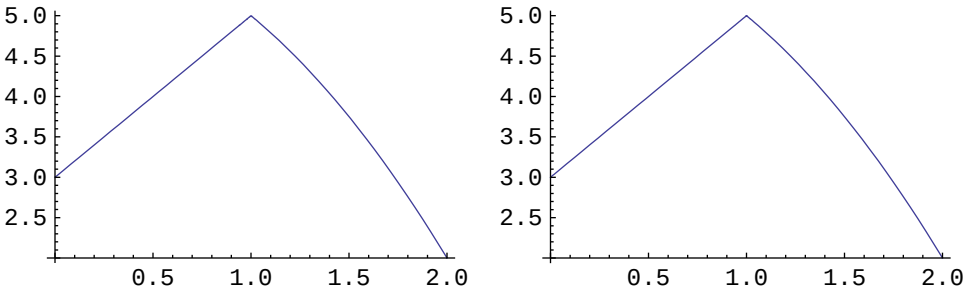
```
f1'[x]
```

```
If[x < 1, 2, -2 x]
```

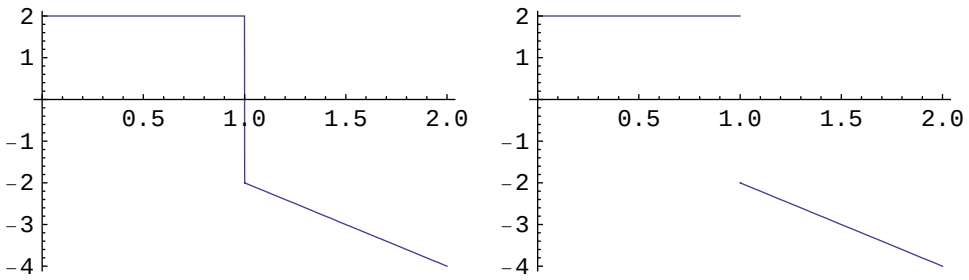
```
f2'[x]
```

```
⎧ 2             x < 1
⎨ -2 x          x > 1
⎩ Indeterminate True
```

```
GraphicsGrid[{{Plot[f1[x], {x, 0, 2}], Plot[f2[x], {x, 0, 2}]}}]
```




```
GraphicsGrid[{{Plot[f1'[x], {x, 0, 2}], Plot[f2'[x], {x, 0, 2}]}}]
```



Allerdings hilft das nicht immer. Hier wird beim Bestimmen der Ableitung in beiden Fällen offensichtlich das boolesche Prädikat zu früh ausgewertet.

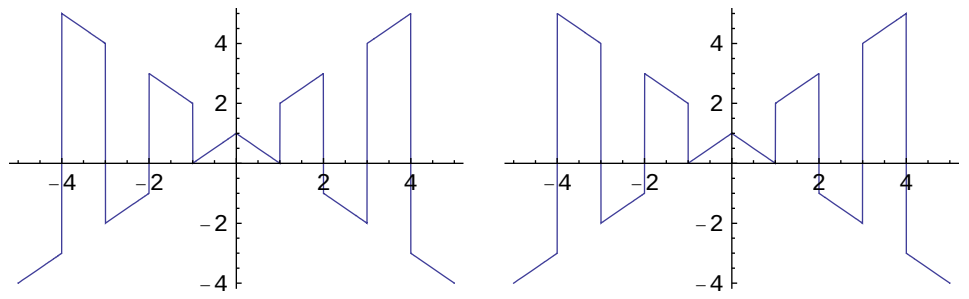
```
g1[x_] := If[OddQ[Floor[x]], x + 1, 1 - x]
g2[x_] := Piecewise[{{x + 1, OddQ[Floor[x]]}}, 1 - x]

{g1'[x], g2'[x]}
```

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

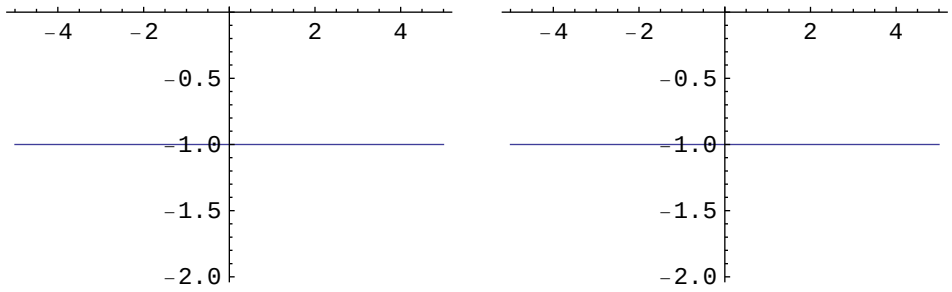
Auch der Graph ist in beiden Varianten eine durchgezogene Linie.

```
GraphicsGrid[{{Plot[g1[x], {x, -5, 5}], Plot[g2[x], {x, -5, 5}]}}]
```



Und selbst die numerische Auswertung ergibt für die Ableitung ein falsches Bild.

```
GraphicsGrid[{{Plot[g1'[x], {x, -5, 5}], Plot[g2'[x], {x, -5, 5}]}}]
```



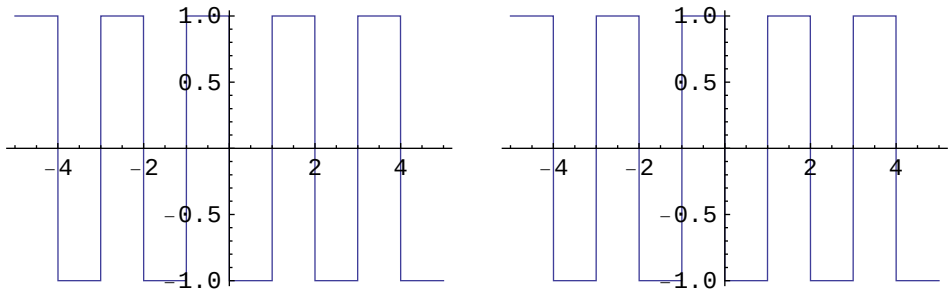
Damit es funktioniert muss **OddQ** durch ein "weicheres" boolesches Prädikat ersetzt werden. Wir verwenden hier eine Version mit Equal (==).

```
g1[x_] := If[Mod[Floor[x], 2] == 1, x + 1, 1 - x]
g2[x_] := Piecewise[{{x + 1, Mod[Floor[x], 2] == 1}}, 1 - x]
```

```
{g1'[x], g2'[x]}
```

$$\left\{\text{If}[\text{Mod}[\text{Floor}[x], 2] == 1, 1, -1], \begin{cases} 1 & \text{Mod}[\text{Floor}[x], 2] == 1 \\ -1 & \text{True} \end{cases}\right\}$$

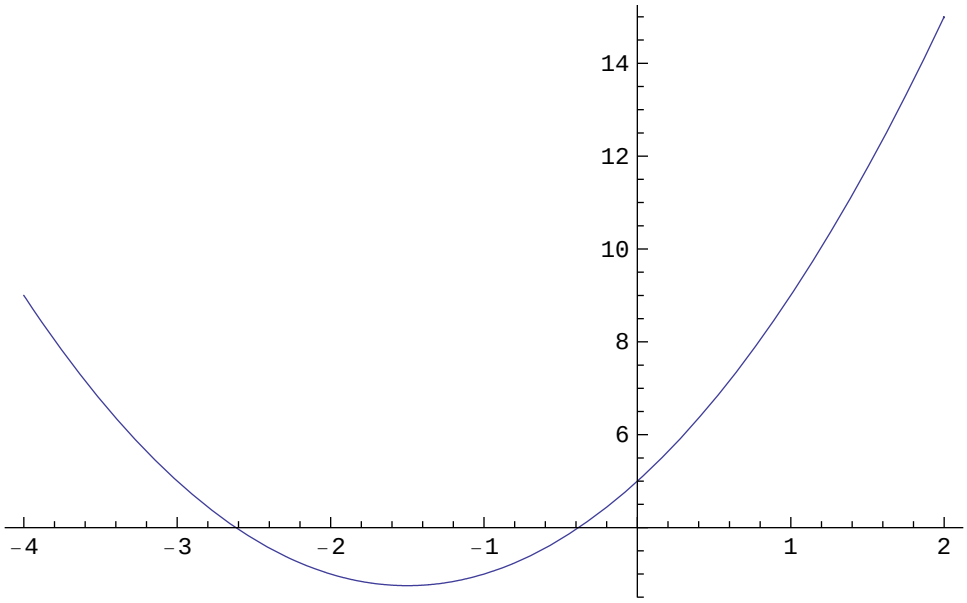
```
GraphicsGrid[{{Plot[g1'[x], {x, -5, 5}], Plot[g2'[x], {x, -5, 5}]}}]
```



■ Funktionsdefinitionen und Blöcke

```
f[x_] := x^2 + 3 x + 5
```

```
Plot[f[x], {x, -4, 2}]
```




```
mySqrt[x_] := Module[{y = x , z = 1},
  While[Abs[y - z] > 10^-5, y = (y + z) / 2.; z = x / y];
  y]
```

```
mySqrt /@ {2, 3, 12, 1221, 21121121}
```

```
{1.41422, 1.73205, 3.4641, 34.9428, 4595.77}
```

```
mySqrt[x]
```

```
x
```

■ Beispiele

▪ Symmetrische Polynome

```
ClearAll["Global`*"]
```

```
e[d_Integer, x_List] := If[Length[x] < d, 0,
  (* else *) If[d == 1, Apply[Plus, x],
   (* else *) Expand[e[d, Rest[x]] + e[d - 1, Rest[x]] * First[x]]]]
```

```
e[5, Table[x_i, {i, 1, 7}]]
```

$$x_1 x_2 x_3 x_4 x_5 + x_1 x_2 x_3 x_4 x_6 + x_1 x_2 x_3 x_5 x_6 + x_1 x_2 x_4 x_5 x_6 + x_1 x_3 x_4 x_5 x_6 + x_2 x_3 x_4 x_5 x_6 + x_1 x_2 x_3 x_4 x_7 + x_1 x_2 x_3 x_5 x_7 + x_1 x_2 x_4 x_5 x_7 + x_1 x_3 x_4 x_5 x_7 + x_2 x_3 x_4 x_5 x_7 + x_1 x_2 x_3 x_6 x_7 + x_1 x_2 x_4 x_6 x_7 + x_1 x_3 x_4 x_6 x_7 + x_2 x_3 x_4 x_6 x_7 + x_1 x_2 x_5 x_6 x_7 + x_1 x_3 x_5 x_6 x_7 + x_2 x_3 x_5 x_6 x_7 + x_1 x_4 x_5 x_6 x_7 + x_2 x_4 x_5 x_6 x_7 + x_3 x_4 x_5 x_6 x_7$$

```
h[d_Integer, x_List] := If[d == 0, 1,
  (* else *) If[Length[x] == 1, x[[1]]^d,
   (* else *) Expand[h[d, Rest[x]] + h[d - 1, x] * First[x]]]]

Table[h[i, {a, b, c}], {i, 0, 3}]
```

$$\{1,\ a+b+c,\ a^2+a\,b+b^2+a\,c+b\,c+c^2,\ a^3+a^2\,b+a\,b^2+b^3+a^2\,c+a\,b\,c+b^2\,c+a\,c^2+b\,c^2+c^3\}$$

▪ Der Euklidsche Algorithmus

```
ClearAll["Global`*"]

Euklid[a0_Integer, b0_Integer] :=
 Module[{a = a0, b = b0, q, r},
  While[b != 0,
   r = Mod[a, b];
   q = (a - r) / b;
   Print[a, " = ", q, "*", b, " + ", r];
   {a, b} = {b, r}
  ]; a]

Euklid[12, 7]
```

```
12 = 1*7 + 5

7 = 1*5 + 2

5 = 2*2 + 1

2 = 2*1 + 0

1
```

```
ExtendedEuklid[a0_Integer, b0_Integer] :=
 Module[{a = a0, b = b0, q, r, ua = 1, ub = 0, va = 0, vb = 1},
  While[b != 0,
   r = Mod[a, b]; q = (a - r) / b;
   {a, b} = {b, r};
   {ua, ub} = {ub, ua - q * ub};
   {va, vb} = {vb, va - q * vb};
   Print[r, " = (", ub, ")*", a0, "+(", vb, ")*", b0, " = ", ub * a0 + vb * b0]
  ]; {a, ua, va}]

ExtendedEuklid[12, 7]
```

```
5 = (1)*12+(-1)*7 = 5

2 = (-1)*12+(2)*7 = 2

1 = (3)*12+(-5)*7 = 1

0 = (-7)*12+(12)*7 = 0

{1, 3, -5}
```

Ein Grafikbeispiel

```
ClearAll["Global`*"]

kreisPunkt[t_] := N[{Cos[180 ° - t], Sin[180 ° - t]}]

w = 230 °;
P2 = kreisPunkt[w]
a = (1 + P2[[1]]) / P2[[2]]
P3 = {-1, a}; P4 = {1, a};
s = 90 °; u = 90 °;
```

```
{0.642788, -0.766044}
```

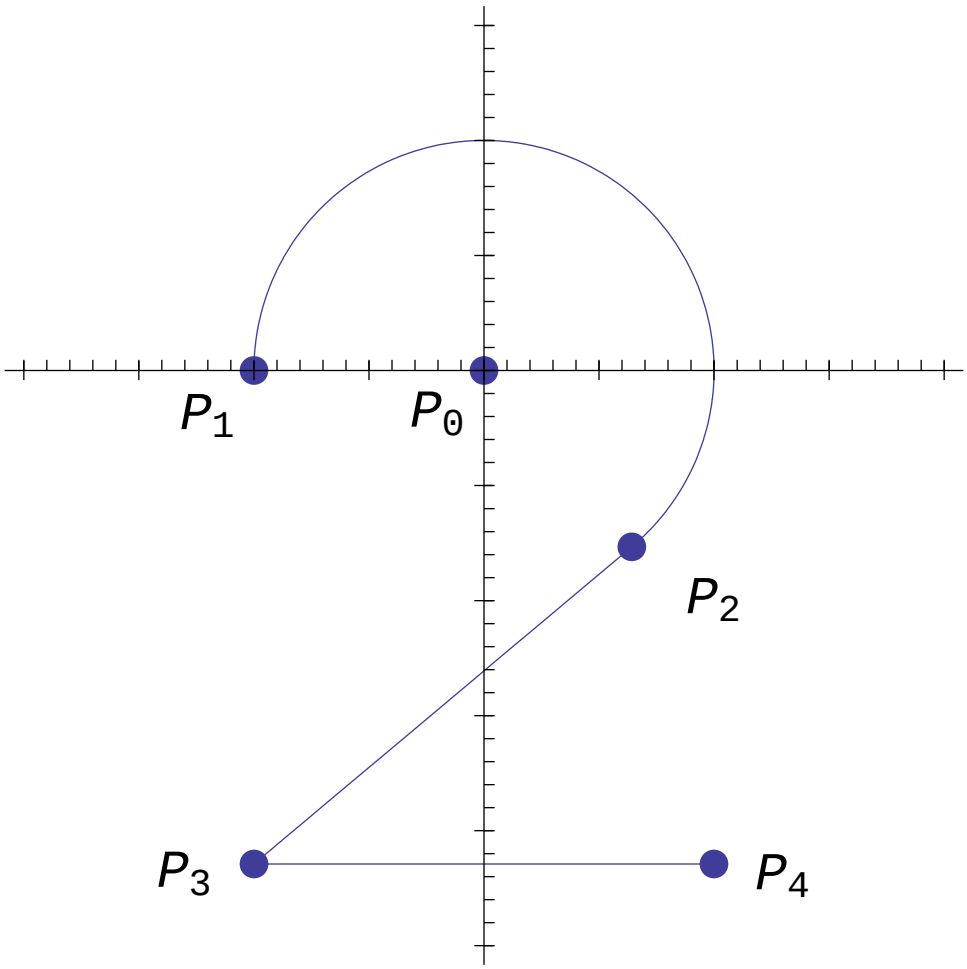
```
-2.14451
```

Die Formel für die stückweise definierte Funktion, mit der das Grundgerüst der Zwei erzeugt wird.

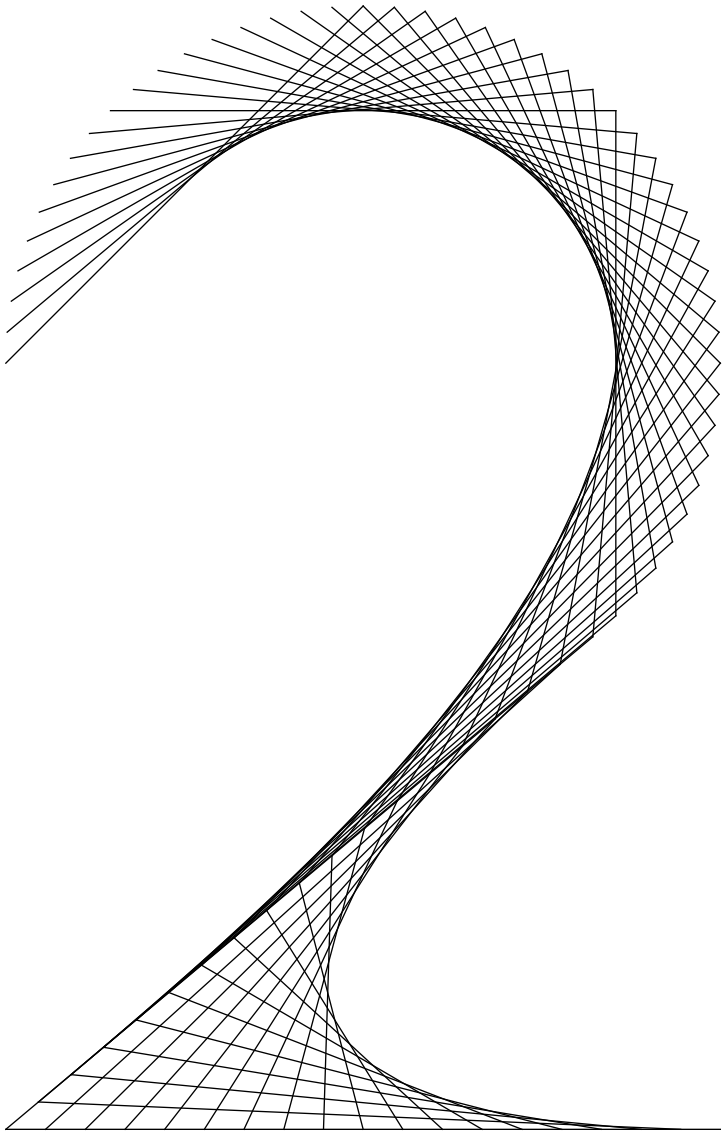
```
P[t_] := Which[
   t ≤ w, kreisPunkt[t],
   t ≤ w + s, (1 - (t - w) / s) P2 + (t - w) / s * P3,
   t ≤ w + s + u, (1 - (t - w - s) / u) P3 + (t - w - s) / u * P4];
```

Hier noch die 5 fetten Punkte, die Ticks auf den Achsen sowie die Beschriftungen als Text−Konstrukte. Die verschiedenen Grafikteile werden mit **Show** gemeinsam angezeigt.

```
fatPoints = {{0, 0}, P[0], P2, P3, P4};
pointLabels = {Text[P0, {-.2, -.2}],
   Text[P1, {-1.2, -0.2}], Text[P2, {1.0, -1.0}], Text[P3, {-1.3, -2.2}],
   Text[P4, {1.3, -2.2}]};
tx = Join[Table[{.1 i,}, {i, -30, 20}],
   Table[{.5 i, , .02}, {i, -5, 10}]];
Show[
 ListPlot[fatPoints, PlotStyle → PointSize[0.03]],
 ParametricPlot[P[t], {t, 0, w + s + u}],
 Graphics[pointLabels],
 PlotRange → {{-2, 2}, {-2.5, 1.5}},
 AspectRatio → 1, Ticks → {tx, tx}, TextStyle → {FontSize → 20}]
```



```
data = Table[Line[{P[t], P[t + s]}], {t, 0, w + u, 5 °}];
Graphics[data]
```



▪ Berechnung der Kettenbruchentwicklung rationaler Zahlen

```
ClearAll["Global‘*"]
```

```
ClearAll[Global‘*]
```

```
Kettenbruch[r0_Rational] :=
 Module[{r = r0, l = {}, v},
  While[! MatchQ[r, _Integer],
   v = Floor[r];
   r = 1 / (r - v);
   l = Append[l, v]];
  Append[l, r]]
```

```
Probe[l_List] := If[Length[l] == 1, First[l], First[l] + 1 / Probe[Rest[l]]]
```

```
Kettenbruch[31 / 245]
```

```
{0, 7, 1, 9, 3}
```

```
% // Probe
```

$$\frac{31}{245}$$

Und hier eine rekursive Version, die aber auch nicht um einen Block herumkommt.

Die erste Variante liefert mysteriöse Ergebnisse:

```
Clear[KRec];
KRec[r_Rational] := Module[{u},
  If[MatchQ[r, _Integer], {r},
   u = Floor[r];
   Prepend[KRec[1 / (r - u)], u]
  ]]
```

```
u = KRec[31 / 245]
```

```
KRec[0, 7, 1, 9, 3]
```

Der Grund ist, dass ganze Zahlen _Rational nicht passieren, so dass KRec[3] symbolisch bleibt.
Zu allem Überfluss funktioniert auch Prepend nicht nur für Listen, sondern ebenso für allgemeinere Ausdrücke.

```
Prepend[u, a]
```

```
KRec[-2.14451, 0, 7, 1, 9, 3]
```

So ist's korrekt:

```
Clear[KRec];
KRec[r_Integer] := {r};
KRec[r_Rational] := Module[{u = Floor[r]}, Prepend[KRec[1 / (r - u)], u]]
```

```
KRec[31 / 245]
```

```
{0, 7, 1, 9, 3}
```

```
% // Probe
```

$$\frac{31}{245}$$

```
ContinuedFraction[31 / 245]
```

```
{0, 7, 1, 9, 3}
```

```
ContinuedFraction[Sqrt[31]]
```

```
{5, {1, 1, 3, 5, 3, 1, 1, 10}}
```

Schnelles modulares Potenzieren

```
ClearAll["Global`*"]

modPower[a0_Integer, n0_Integer, m_Integer] :=
 Module[{a = a0, n = n0, p = 1},
  While[n > 0,
   Print[p, "*", a, "^", n, " = "];
   If[OddQ[n], p = Mod[a * p, m]; n -= 1];
   a = Mod[a * a, m]; n /= 2];
  p]

modPower[2, 107, 163]
```

```
1*2^107 =

2*4^53 =

8*16^26 =

8*93^13 =

92*10^6 =

92*100^3 =

72*57^1 =

29
```

```
PowerMod[2, 107, 163]
```

```
29
```

Fibonaccizahlen

```
ClearAll["Global`*"]

fib[n_Integer] := Switch[n, 0, 0, 1, 1, _, fib[n - 1] + fib[n - 2]]

?fib
```

```
Global`fib
```

```
fib[n_Integer] := Switch[n,
  0, 0,
  1, 1,
  _, fib[n - 1] + fib[n - 2]]
```

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

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

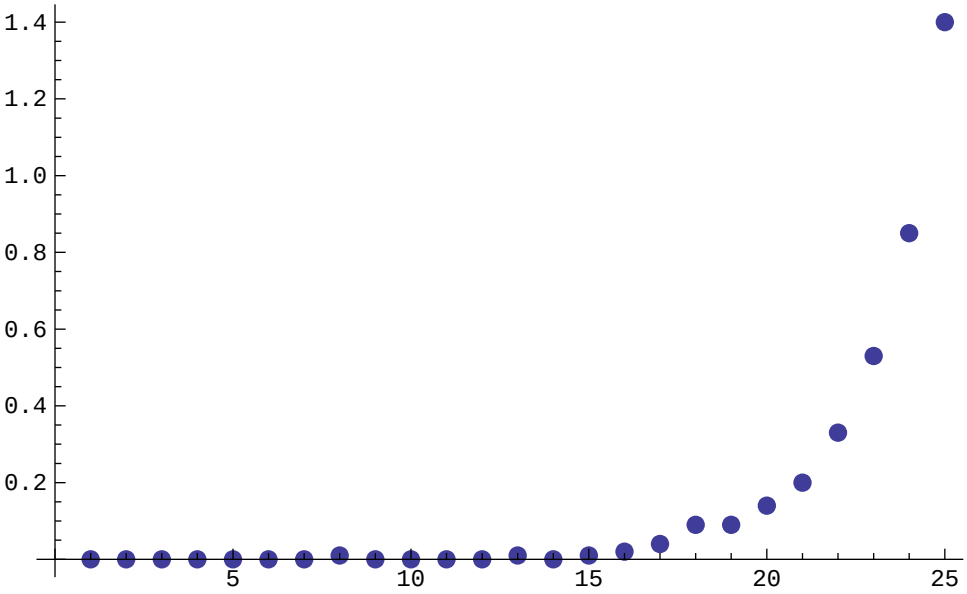
```
Timing[fib[25]]
```

```
{1.45, 75 025}
```

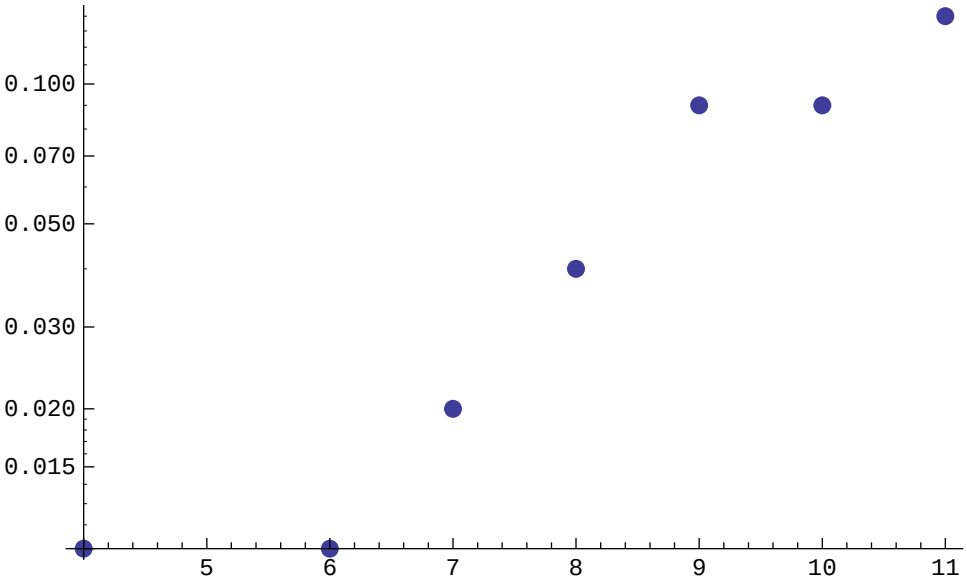
```
l = Table[Timing[fib[i]][[1]] // Chop, {i, 25}]
```

```
{0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0.01, 0, 0.01,
 0.02, 0.04, 0.09, 0.09, 0.14, 0.2, 0.33, 0.53, 0.85, 1.4}
```

```
ListPlot[l, PlotStyle → PointSize[0.02], PlotRange -> All , AxesOrigin → {0, 0}]
```




```
l1 = Table[l[[i]], {i, 10, 20}];
ListLogPlot[l1, PlotStyle → PointSize[0.02]]
```




```
Clear[fib];
fib[n_Integer] := fib[n] = Switch[n, 0, 0, 1, 1, _, fib[n - 1] + fib[n - 2]]
```

```
?fib
```

```
Global`fib
```

```
fib[n_Integer] := fib[n] = Switch[n,
   0, 0,
   1, 1,
   _, fib[n - 1] + fib[n - 2]]
```

Achtung, **$RecursionLimit** wird jetzt bei jedem rekursiven Aufruf hochgezählt und nicht nur bei verschachtelten.

```
fib[20]
```

```
6765
```

```
?fib
```

```
Global`fib
```

```
fib[0] = 0

fib[1] = 1

fib[2] = 1

fib[3] = 2

fib[4] = 3

fib[5] = 5

fib[6] = 8

fib[7] = 13

fib[8] = 21

fib[9] = 34

fib[10] = 55

fib[11] = 89

fib[12] = 144

fib[13] = 233

fib[14] = 377

fib[15] = 610

fib[16] = 987

fib[17] = 1597

fib[18] = 2584

fib[19] = 4181

fib[20] = 6765

fib[n_Integer] := fib[n] = Switch[n,
   0, 0,
   1, 1,
   _, fib[n - 1] + fib[n - 2]]
```

Auch die Größe der Fibonaccizahlen wächst exponentiell, so dass wir uns in den folgenden Beispielen auf die Berechnung der jeweils letzten 20 Ziffern beschränken.

```
Clear[fib];
fib[n_Integer] := Module[{a = 0, b = 1}, Do[{a, b} = {b, Mod[a + b, 10^20]}, {n}]; a]
```

```
Table[fib[i], {i, 20}]
```

```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765}
```

```
?fib
```

```
Global`fib
```

```
fib[n_Integer] := Module[{a = 0, b = 1}, Do[{a, b} = {b, Mod[a + b, 10^20]}, {n}]; a]
```

Diese Definition hat nicht die Laufzeitprobleme der zuerst gegebenen rekursiven Definition und auch nicht die Speicherplatzprobleme der zweiten.

```
Timing[fib[1000]]
```

```
{0.01, 76 137 795 166 849 228 875}
```

Und so geht es noch schneller (obwohl **MatrixPower** nicht mit modularem Rechnen gekoppelt werden kann):

```
matfib[n_] := (MatrixPower[({{1, 1}, {1, 0}}, n] . {1, 0})[[2, 1]]
```

```
matfib[#] & /@ Range[20]
```

```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765}
```

```
n = 5;
First /@ {fib[10^n] // Timing, matfib[10^n] // Timing}
```

```
{1.03, 0.01}
```

```
l = Select[Range[10^3], (Mod[fib[#], 10^2] == 99) &]
```

```
{52, 298, 352, 598, 652, 898, 952}
```

```
fib /@ l
```

```
{32 951 280 099, 63 883 732 297 726 369 399, 50 176 124 155 306 760 699, 13 614 497 857 812 538 799,
 71 110 319 794 282 041 299, 50 341 132 602 138 508 199, 10 659 067 317 997 121 899}
```