

Kurs ■ Cours

9. Funktionen definieren ■ Définir des fonctions

Die Gliederung dieses Kurses folgt in groben Zügen dem Buch von Nancy Blachman: A Practical Approach....

Hinweis: Kapitel 9 lesen!

Run mit WIN+*Mathematica* Version 5.2

■ L'articulation de ce cours correspond à peu près à celle du livre de Nancy Blachman: A Practical Approach....

Indication: Lire le chapitre 9.

Testé avec *Mathematica* version 5.2+WIN

WIR94/98/99/2000/2007 // Copyright Rolf Wirz

9.1. Einfache Funktionen

■ Fonctions simples

9.1.1. Funktionen selber schreiben: Falls in *Mathematica* die gewollte Funktion nicht schon vorhanden ist, kann man versuchen, sie selber zu schreiben.

■ Ecrire les fonctions soi-même: Si la fonction voulue n'est pas contenue dans *Mathematica*, on peut l'écrire soi-même.

Erst "Clear", damit keine Ueberraschung auftritt! Beispiel: Quadrieren...

■ D'abord "Clear", pour ne pas avoir de susprise! Exemple: Elever au carré...

```
In[1]:= Clear[quadrat];  
        quadrat[x]:=x^2;  
        quadrat[x]
```

```
Out[3]= x2
```

```
In[4]:= {quadrat[y], quadrat[5], quadrat[12]}
```

```
Out[4]= {quadrat[y], quadrat[5], quadrat[12]}
```

Offenbar klappt es bei dieser Funktion nicht mit dem Variablenwechsel resp. mit dem Einsetzen von Werten. Wieso?

■ Il est évident que le changement de variables ne fonctionne pas pour cette fonction, le placement de valeurs non plus. Pourquoi?

```
In[5]:= ??quadrat
```

```
Global`quadrat
```

```
quadrat[x] := x2
```

```
In[6]:= ?_
```

```
_ or Blank[ ] is a pattern object that can stand for any Mathematica  
expression. _h or Blank[h] can stand for any expression with head h. Mehr...
```

Die Variable x ist kein "Muster", das für irgend einen andern Ausdruck oder Wert stehen kann. Nehmen wir "x_" statt "x", so klappt es:

■ La variable x n'est pas un "patron" qui peut remplacer une expression ou une valeur quelconque. Si nous prenons "x_" au lieu de "x", cela fonctionne:

```
In[7]:= Clear[quadrat1];  
quadrat1[x_] := x2;  
quadrat1[x]
```

```
Out[9]= x2
```

```
In[10]:= {quadrat1[y], quadrat1[5], quadrat1[12]}
```

```
Out[10]= {y2, 25, 144}
```

```
In[11]:= ??quadrat1
```

```
Global`quadrat1
```

```
quadrat1[x_] := x2
```

Oder: ■ Ou:

```
In[12]:= Table[quadrat1[n], {n, 12}]
```

```
Out[12]= {1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144}
```

Oder: ■ Ou:

```
In[13]:= quadrat1[9z3+2z-1]
```

```
Out[13]= (-1 + 2 z + 9 z3)2
```

```
In[14]:= quadrat1[9z3+2z-1] // Expand
```

```
Out[14]= 1 - 4 z + 4 z2 - 18 z3 + 36 z4 + 81 z6
```

Oder: ■ Ou:

9.1.2. Funktionen mit mehreren Variablen:

■ Fonction avec plusieurs variables:

Ein Beispiel, dass alles sagt (Cosinus-Satz):

■ Un exemple qui dit tout (théorème du cosinus):

```
In[15]:= Clear[c];
          c[a_, b_, gamma_] = Sqrt[a^2 + b^2
                                - 2 a b Cos[gamma]];
          c[u, v, w]
```

```
Out[17]=  $\sqrt{u^2 + v^2 - 2 u v \cos[w]}$ 
```

```
In[18]:= c[3, 5, Pi/3]
```

```
Out[18]=  $\sqrt{19}$ 
```

9.1.3. Funktionen definiert in mehreren Schritten:

■ Fonctions définies en plusieurs étapes:

Beispiel: Umrechnung in Polarkoordinaten:

■ Exemple: Transformation en coordonnées polaires:

```
In[19]:= PolarTransformation[r_, phi_] := (
          x = r Cos[phi];
          y = r Sin[phi];
          {x, y}
        );
        PolarTransformation[a, b]
```

```
Out[20]= {a Cos[b], a Sin[b]}
```

Neudefinition: Die Umrechnung in Polarkoordinaten könnte auch für einen Vektor aufgerufen werden. Daher ist eine zweite Definition ratsam:

■ Nouvelle définition: La transformation en coordonnées polaires pourrait être appelée aussi pour un vecteur. C'est pourquoi on conseille une deuxième définition:

```
In[21]:= PolarTransformation[{r_, phi_}] :=
          PolarTransformation[r, phi];
          PolarTransformation[{a, b}]
```

```
Out[22]= {a Cos[b], a Sin[b]}
```

```
In[23]:= PolarTransformation[{3, 2}]
```

```
Out[23]= {3 Cos[2], 3 Sin[2]}
```

```
In[24]:= PolarTransformation[3, 2]
```

```
Out[24]= {3 Cos[2], 3 Sin[2]}
```

```
In[25]:= PolarTransformation[3, 2] // N
```

```
Out[25]= {-1.24844, 2.72789}
```

9.2. Typenprüfung

■ Examen de types

9.2.1. f[arg_typ] ■ f[arg_typ]

"f[arg_typ]" funktioniert nur, falls gilt "Head[arg] === typ". Probiere:

■

"f[arg_typ]" fonctionne seulement s'il vaut "Head[arg] === typ". Essaie:

```
In[26]:= ?Binomial
```

Binomial[n, m] gives the binomial coefficient. Mehr...

```
In[27]:= Clear[pascal];
          pascal[n_Integer]:=Table[Binomial[n,i],{i,0,n}];
```

```
In[29]:= pascal[0]
```

```
Out[29]= {1}
```

```
In[30]:= pascal[1]
```

```
Out[30]= {1, 1}
```

```
In[31]:= pascal[2]
```

```
Out[31]= {1, 2, 1}
```

```
In[32]:= pascal[3]
```

```
Out[32]= {1, 3, 3, 1}
```

```
In[33]:= pascal[4]
```

```
Out[33]= {1, 4, 6, 4, 1}
```

```
In[34]:= pascal[5]
```

```
Out[34]= {1, 5, 10, 10, 5, 1}
```

```
In[35]:= pascal[5.6]
```

```
Out[35]= pascal[5.6]
```

```
In[36]:= pascal[6/5]
```

```
Out[36]= pascal[ $\frac{6}{5}$ ]
```

Ein anderes Beispiel (ausprobieren):

■ Autre exemple (essayer):

```
In[37]:= Clear[mittelWert];
```

```
In[38]:= mittelWert[x_List] := Apply[Plus, x]/Length[x];
        mittelWert[{1,2,3,4,5}]
```

```
Out[39]= 3
```

```
In[40]:= mittelWert[1,2,3,4,5]
```

```
Out[40]= mittelWert[1, 2, 3, 4, 5]
```

Test, ob ein Ausdruck ein Polynom, eine Primzahl, ein Vektor u.s.w. ist, d.h. ob ein Ausdruck unter einer Prädikatenfunktion "wahr" ergibt. Beispiel:

■ Tester si une expression est un polynôme, un nombre premier, un vecteur etc., c'est-à-dire, si une expression est "vraie" sous une fonction de prédicats. Exemple:

```
In[41]:= ?*Q
```

System`

ArgumentCountQ	LinkConnectedQ	PartitionsQ
ArrayQ	LinkReadyQ	PolynomialQ
AtomQ	ListQ	PrimeQ
DigitQ	LowerCaseQ	SameQ
EllipticNomeQ	MachineNumberQ	StringFreeQ
EvenQ	MatchLocalNameQ	StringMatchQ
ExactNumberQ	MatchQ	StringQ
FreeQ	MatrixQ	SyntaxQ
HypergeometricPFQ	MemberQ	TensorQ
InexactNumberQ	NameQ	TrueQ
IntegerQ	NumberQ	UnsameQ
IntervalMemberQ	NumericQ	UpperCaseQ
InverseEllipticNomeQ	OddQ	ValueQ
LegendreQ	OptionQ	VectorQ
LetterQ	OrderedQ	

```
In[42]:= {PrimeQ[45], PrimeQ[17]}
```

```
Out[42]= {False, True}
```

Anwendung auf die Definition einer Funktion:

■ Application à la définition d'une fonction:

```
In[43]:= Clear[halb];
        halb[x_?EvenQ] := x/2;
        {halb[1], halb[2], halb[3], halb[4], halb[5]}
```

```
Out[45]= {halb[1], 1, halb[3], 2, halb[5]}
```

```
In[46]:= {halb[2], halb[4], halb[6], halb[8], halb[9]}
```

```
Out[46]= {1, 2, 3, 4, halb[9]}
```

Was ist passiert?

■ Que s'est-il passé?

Ein Beispiel in mehreren Schritten:

■ Un exemple en plusieurs étapes:

```
In[47]:= Remove[signum];
          signum[0] := 0;
          signum[x_ /; Positive[x]==True] := 1;
          signum[x_ /; Negative[x]==True] := -1;

In[51]:= {signum[3], signum[-1], signum[0], signum[2]}

Out[51]= {1, -1, 0, 1}

In[52]:= ??Positive

Positive[x] gives True if x is a positive number. Mehr...

Attributes[Positive] = {Listable, Protected}

In[53]:= ??Integer

Integer is the head used for integers. Mehr...

Attributes[Integer] = {Protected}

In[54]:= Positive[1]

Out[54]= True

In[55]:= signum[1]

Out[55]= 1

In[56]:= ??signum

Global`signum

signum[0] := 0

signum[x_ /; Positive[x] == True] := 1

signum[x_ /; Negative[x] == True] := -1
```

9.3. Bedingte Ausführung

■ Exécution conditionnée

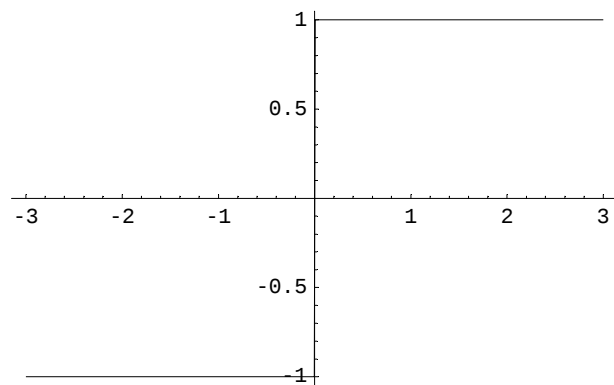
Betrachte das obige Beispiel mit "signum"!

■ Considère l'exemple ci-dessus avec "signum"!

Bei der Definition von "signum" ist in der Klammer "[]" nach "x_" das Symbol "/" angegeben. Darauf folgt jeweils eine Bedingung! Damit können Funktionen punktweise oder intervallweise definiert werden!!!! Anwendungen:

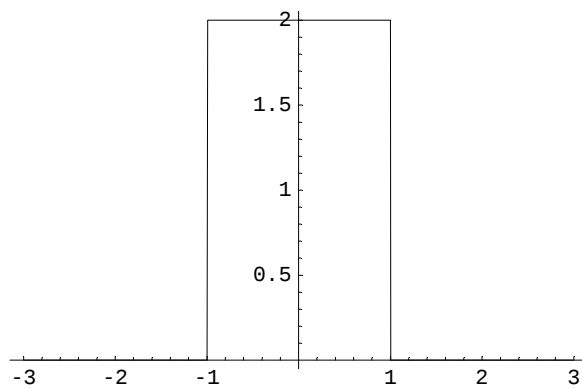
■ Dans la définition de "signum", le symbole "/" est indiqué dans la parenthèse "[]" après "x_". Il suit chaque fois une condition! Ainsi on peut définir des fonctions ponctuellement ou par intervalles! Applications:

```
In[57]:= Plot[signum[x], {x, -3, 3}];
```



Oder: ■ Ou:

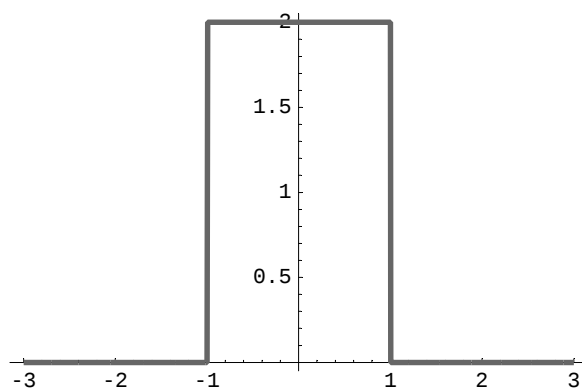
```
In[58]:= Clear[rechteck];
rechteck[x_ /; (-1 <= x && x <= 1)] := 2;
rechteck[x_ /; Abs[x] > 1] := 0;
Plot[rechteck[x], {x, -3, 3}];
```



Etwas "schöner":

■ Un peu plus esthétiquement:

```
In[62]:= Plot[rechteck[x], {x, -3, 3},
  PlotStyle -> {{Thickness[0.01],
    GrayLevel[0.4]}}];
```



Solche Funktionen können sogar integriert werden:

■ On peut même intégrer de telles fonctions:

In[63]:= ??rechteck

Global`rechteck

rechteck[x_ /; $-1 \leq x \leq 1$] := 2

rechteck[x_ /; Abs[x] > 1] := 0

In[64]:= NIntegrate[rechteck[x], {x, -1, 1}]

Out[64]= 4.

In[65]:= NIntegrate[rechteck[x], {x, -3, 3}]

NIntegrate::slwcon :

Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration being 0, oscillatory integrand, or insufficient WorkingPrecision. If your integrand is oscillatory try using the option Method->Oscillatory in NIntegrate. Mehr...

NIntegrate::ncvb : NIntegrate failed to converge to

prescribed accuracy after 7 recursive bisections in x near x = -1.00781. Mehr...

Out[65]= 3.99241

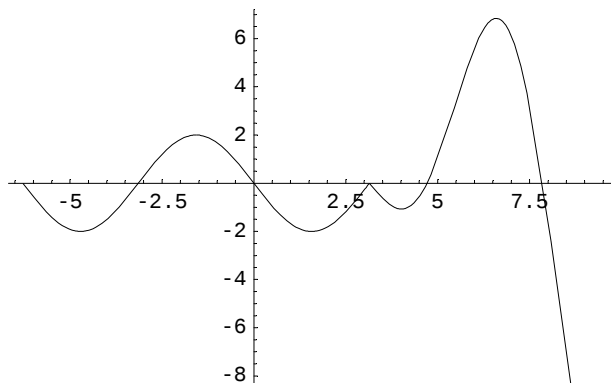
Was ist passiert? Studiere:

■ Que s'est-il passé? Etudie:

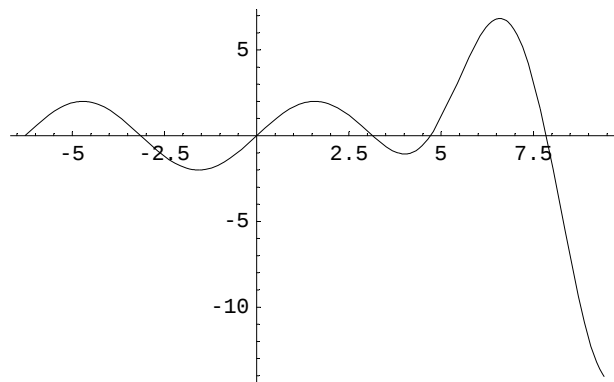
In[66]:= ?Which

Which[test1, value1, test2, value2, ...] evaluates each of the testi in turn, returning the value of the valuei corresponding to the first one that yields True. Mehr...

In[67]:= Remove[funkt];
funkt[x_] := Which[x<=Pi, -2 Sin[x], x>Pi,
(x^1.3-Pi^1.3) Cos[x]];
Plot[funkt[x], {x, -2Pi, 3Pi}];



```
In[70]:= Remove[funkt];
funkt[x_] := Which[x<=Pi, 2 Sin[x], x>Pi,
  (x^1.3-Pi^1.3) Cos[x]];
Plot[funkt[x], {x, -2Pi, 3Pi}];
```



9.4. Vorgegebene Werte

■ Valeurs données

9.4.1. Das Muster "x_.":

■ Le patron ("Pattern") "x_.":

Das Muster "x_" bewirkt, dass x "default" mit dem neutralen Element der am "stärksten" mit x verknüpften Operation belegt wird. Dazu ein Beispiel:

■ Le patron "x_" fait que x "default" est occupé par l'élément neutre de l'opération qui est le plus "fortement" lié à x. Voici un exemple:

```
In[73]:= Clear[f];
f[x_. + y_. z_.^e_.] := {x, y, z, e};
```

Pattern::nodef : No default setting found for Power in position 1 when length is 2. Mehr...

Pattern::nodef : No default setting found for Power in position 1 when length is 2. Mehr...

```
In[75]:= Clear[f];
f[x_. + y_. z_.^e_.] := {x, y, z, e};
```

```
In[77]:= ?f
```

Global`f

```
f[x_. + y_. z_.^e_.] := {x, y, z, e}
```

```
In[78]:= f[4]
```

```
Out[78]= {0, 1, 4, 1}
```

```
In[79]:= {f[0], f[1], f[2], f[3], f[4], f[5]}
```

```
Out[79]= {{0, 1, 0, 1}, {0, 1, 1, 1}, {0, 1, 2, 1}, {0, 1, 3, 1}, {0, 1, 4, 1}, {0, 1, 5, 1}}
```

```
In[80]:= {x, y, z, e}
```

```
Out[80]= {3 Cos[2], 3 Sin[2], z, e}
```

Finde heraus, welche Werte "default" jeweils für x, y, z und e genommen worden sind und wieso!

■ Cherche quelles valeurs "default" ont été prises respectivement pour x, y, z et e et pour quelle raison!

9.4.2. Das Muster "x_:" :

■ Le patron "x_:" :

Das Muster "x_:Wert" bewirkt, dass x "default" mit Wert belegt wird. Dazu ein Beispiel:

■ Le patron "x_:Valeur" fait que x "default" est occupé par valeur. Voici un exemple:

```
In[81]:= Clear[UnserBereich];
         UnserBereich[start_:34, ziel_] :=
           Range[start, ziel];
         UnserBereich[100]
```

```
Out[83]= {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}
```

```
In[84]:= UnserBereich[80, 100]
```

```
Out[84]= {80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100}
```

```
In[85]:= UnserBereich[36]
```

```
Out[85]= {34, 35, 36}
```

```
In[86]:= UnserBereich[34]
```

```
Out[86]= {34}
```

```
In[87]:= UnserBereich[30]
```

```
Out[87]= {}
```

9.5. Die Abarbeitungshierarchie bei Regeln

Mathematica bearbeitet die spezielleren Regeln vor den generelleren Regeln. Mit "?" kann die Abarbeitungshierarchie eingesehen werden.

■ *Mathematica* travaille les règles spéciales avant les règles générales. Avec "?" on peut voir la hiérarchie de travail.

```
In[88]:= Clear[r];
         r[x_]      := x^2;
         r[x_Integer] := x;
         r[5]       := 11;
         ?r

Global`r

r[5] := 11

r[x_Integer] := x

r[x_] := x^2
```

Beachte, dass die Reihenfolge der Ausgabe nicht mit der der Eingabe übereinstimmt. Wieso wohl?
 Considère que l'ordre de la sortie ne correspond pas à celui de l'entrée. Pourquoi?

```
In[93]:= {r[5], r[5.5], r[6], r[r.5], r[7]}

Out[93]= {11, 30.25, 6, 0.25 r^2, 7}
```

9.6. Zusammenfügen von Regeln mit "f/: "

■ Assemblage de règles par "f/: "

Operationen links vom Zeichen ":= "

■ Opérations à gauche du signe ":= "

Z.B. $f[x_] + g[x_] := h[x_]$ "funktioniert" nicht. Beispiel:
 ■ P.ex. $f[x_] + g[x_] := h[x_]$ ne "fonctionne" pas. Exemple:

```
In[94]:= f[x_] + g[x_] := h[x_]

SetDelayed::write : Tag Plus in g[x_] + {0, 1, x_, 1} is Protected. Mehr...

Out[94]= $Failed
```

Wie lässt sich das Funktionieren trotzdem erzwingen? -
 Dafür ist das Symbol "f/: " vorgesehen. Studiere das folgende Beispiel:

■ Comment obtenir quand même le fonctionnement? -
 Pour cela est prévu le symbole "f/: ". Etudie l'exemple suivant:

```
In[95]:= Remove[f];
         f/: f[x_] + g[x_] := h[x_]

RuleDelayed::rhs : Pattern x_ appears on the right-hand side of rule f[x_] + g[x_] => h[x_]. Mehr...

In[97]:= ?f

Global`f

f/: f[x_] + g[x_] := h[x_]

In[98]:= f[5] + g[5]

Out[98]= h[Pattern[5, _]]
```

f wird hier assoziiert mit $f[x]+g[x] := h[x]$. Wird $f[x]+g[x]$ gerufen, so erscheint $h[x]$. Andernfalls erscheint $f[x]$ oder $f[x1]+g[x2]$.

■ Ici on associe f avec $f[x]+g[x] := h[x]$. Si on appelle $f[x]+g[x]$, il arrive $h[x]$. Autrement il apparaît $f[x]$ ou $f[x1]+g[x2]$.

```
In[99]:= f[5]
```

```
Out[99]= f[5]
```

```
In[100]:=
```

```
    f[5] + g[6]
```

```
Out[100]=
```

```
    f[5] + g[6]
```

Beispiel: ■ Exemple:

```
In[101]:=
```

```
    Remove[f];
```

```
    f/: f[x_] + Sin[x_] := x^3;
```

```
    f[3] + Sin[3]
```

```
Out[103]=
```

```
    27
```

Vergleiche: ■ Compare:

```
In[104]:=
```

```
    ??_
```

_ or Blank[] is a pattern object that can stand for any Mathematica expression. _h or Blank[h] can stand for any expression with head h. Mehr...

```
Attributes[Blank] = {Protected}
```

```
In[105]:=
```

```
    ?_.
```

p:v is a pattern object which represents an expression of the form p, which, if omitted, should be replaced by v. Mehr...

```
In[106]:=
```

```
    ?:
```

```
Information::notfound : Symbol : not found. Mehr...
```

```
In[107]:=
```

```
    ?/;
```

patt /; test is a pattern which matches only if the evaluation of test yields True. lhs :=> rhs /; test represents a rule which applies only if the evaluation of test yields True. lhs := rhs /; test is a definition to be used only if test yields True. Mehr...

```
In[108]:=
```

```
    ???
```

p?test is a pattern object that stands for any expression which matches p, and on which the application of test gives True. Mehr...

```
Attributes[PatternTest] = {HoldRest, Protected}
```

9.7. Dokumentieren der eigenen Funktionen

■ Documenter les propres fonctions

9.7.1. Verschiedenes Verhalten einer Transformationsregel:

■ Comportement différent d'une règle de transformation

Beachte, dass die Regel $s[-x_] \rightarrow -s[x]$ den Ausdruck $s[-a]$ in $-s[a]$ transformiert, $s[-3]$ aber nicht transformiert. Was wird mit $s[1-x]$ geschehen und wieso?

■ Considère que la règle $s[-x_] \rightarrow -s[x]$ transforme l'expression $s[-a]$ en $-s[a]$, mais ne transforme pas $s[-3]$. Que se passe-t-il avec $s[1-x]$ et pourquoi?

```
In[109]:=
  sRegel = s[-x_] -> -s[x]
```

```
Out[109]=
  s[-x_] -> -s[x]
```

```
In[110]:=
  s[-a] /. sRegel
```

```
Out[110]=
  -s[a]
```

```
In[111]:=
  s[-3] /. sRegel
```

```
Out[111]=
  s[-3]
```

Wieso geschieht das? Beobachte, wie *Mathematica* -a und -3 interpretiert:

■ Pourquoi cela arrive-t-il? Observe comment *Mathematica* interprète -a et -3:

```
In[112]:=
  FullForm[-a]
```

```
Out[112]//FullForm=
  Times[-1, a]
```

```
In[113]:=
  FullForm[-3]
```

```
Out[113]//FullForm=
  -3
```

Der Ausdruck -a ist mit dem Muster $-(x_)$ verträglich, der Ausdruck -3 nicht.

■ L'expression -a est compatible avec le patron $-(x_)$, l'expression -3 ne l'est pas.

```
In[114]:=
  FullForm[1-x]

Out[114]//FullForm=
  Plus[1, Times[-3, Cos[2]]]
```

Der Ausdruck (1-x) ist mit dem Muster -(x_) nicht verträglich:

■ L'expression (1-x) n'est pas compatible avec le patron -(x_).

```
In[115]:=
  s[1-x] /. sRegel

Out[115]=
  s[1 - 3 Cos[2]]
```

9.7.2. Der Ausdruck ' Funktion ::usage = "Erklärung" '

■ L'expression ' Funktion ::usage = "explication" '

Bisher ist öfters schon das Symbol "?" erschienen. Damit ist einige Information abrufbar. Beispiel:

■ Jusqu'à présent le symbole "?" est apparu plusieurs fois. Par cela on peut appeler bien de l'information. Exemple:

```
In[116]:=
  ?f

Global`f

f /: f[x_] + Sin[x_] := x^3

In[117]:=
  ?Sin

Sin[z] gives the sine of z. Mehr...

In[118]:=
  ??Sin

Sin[z] gives the sine of z. Mehr...

Attributes[Sin] = {Listable, NumericFunction, Protected}
```

Manchmal möchte man aber gern dem System eigene Kommentare einverleiben, die dann wieder abgerufen werden können. Das ist möglich mit "....::usage=...".

Beispiel: Wir verwenden die oben definierte Funktion "pascal".

■ Parfois on aimerait incorporer au système ses propres commentaires, qu'on pourrait de nouveau appeler. Cela est possible avec "....::usage=...".

Exemple: Nous employons la fonction "pascal" définie ci-dessus.

```
In[119]:=
  Clear[pascal];
  pascal[n_Integer]:=Table[Binomial[n,i],{i,0,n}];

In[121]:=
  ?pascal

Global`pascal

pascal[n_Integer] := Table[Binomial[n, i], {i, 0, n}]
```

Wir fügen nun an:

■ Nous ajoutons:

```
In[122]:=
    pascal::usage = "pascal[n] gibt die n-te Zeile
    des Pascalschen Dreiecks aus - sort la ligne n
    du triangle de Pascal."

Out[122]=
    pascal[n] gibt die n-te Zeile des Pascalschen
    Dreiecks aus - sort la ligne n du triangle de Pascal.
```

Abfragen: ■ Appeler:

```
In[123]:=
    ?pascal

    pascal[n] gibt die n-te Zeile des
    Pascalschen Dreiecks aus - sort la ligne n du triangle de Pascal.

In[124]:=
    ??pascal

    pascal[n] gibt die n-te Zeile des
    Pascalschen Dreiecks aus - sort la ligne n du triangle de Pascal.

    pascal[n_Integer] := Table[Binomial[n, i], {i, 0, n}]
```

Gute Sache, nicht?

■ C'est bon, n'est-ce pas?

9.8. Attribute

■ Attributs

Spezielle Funktionen bestimmen:

■ Déterminer des fonctions spéciales:

Funktionen haben Attribute. Beispiel "Sinus":

■ Les fonctions ont des attributs. Exemple "Sinus":

```
In[125]:=
    Attributes[Sin]

Out[125]=
    {Listable, NumericFunction, Protected}

In[126]:=
    ??Attributes

    Attributes[symbol] gives the list of attributes for a symbol. Mehr...

    Attributes[Attributes] = {HoldAll, Listable, Protected}
```

In[127]:=

??Attributes

Attributes[symbol] gives the list of attributes for a symbol. Mehr...

Attributes[Attributes] = {HoldAll, Listable, Protected}

Die möglichen Attribute sind vom System vorgegeben. Einige Beispiele:

■ Les attributs possibles sont donnés par le système. Quelques exemples:

In[128]:=

??Constant

Constant is an attribute which indicates
zero derivative of a symbol with respect to all parameters. Mehr...

Attributes[Constant] = {Protected}

In[129]:=

?Flat

Flat is an attribute that can be assigned to a symbol f to
indicate that all expressions involving nested functions f should be
flattened out. This property is accounted for in pattern matching. Mehr...

In[130]:=

?Hold*

System`

Hold	HoldAllComplete	HoldFirst	HoldPattern
HoldAll	HoldComplete	HoldForm	HoldRest

In[131]:=

?HoldAll

HoldAll is an attribute which specifies that all arguments
to a function are to be maintained in an unevaluated form. Mehr...

In[132]:=

?HoldFirst

HoldFirst is an attribute which specifies that the first
argument to a function is to be maintained in an unevaluated form. Mehr...

In[133]:=

?HoldRest

HoldRest is an attribute which specifies that all but the first
argument to a function are to be maintained in an unevaluated form. Mehr...

In[134]:=

?Listable

Listable is an attribute that can be assigned to a symbol f to indicate that the function f
should automatically be threaded over lists that appear as its arguments. Mehr...

In[135]:=

?Locked

Locked is an attribute which, once assigned,
prevents modification of any attributes of a symbol. Mehr...

In[136]:=

?Orderless

Orderless is an attribute that can be assigned to a symbol *f* to indicate that the elements *ei* in expressions of the form *f*[*e1*, *e2*, ...] should automatically be sorted into canonical order. This property is accounted for in pattern matching. Mehr...

In[137]:=

?Protected

Protected is an attribute which prevents any values associated with a symbol from being modified. Mehr...

In[138]:=

?ReadProtected

ReadProtected is an attribute which prevents values associated with a symbol from being seen. Mehr...

9.9. Die Art der Abarbeitung eines Ausdrucks

■ La façon de travailler une expression

9.9.1. Normalfall

■ Cas normal

Folgende Reihenfolge ist im System voreingestellt: Zuerst Kopf des Ausdrucks abarbeiten. Dann Elemente (Teile) des Ausdrucks abarbeiten.

Wenn ein Element kein Atomarer Ausdruck ist: Teile des Elements abarbeiten etc.. Dieses Procedere wird rekursiv angewandt.

■ L'ordre suivant est déjà réglé dans le système: Travailler d'abord l'en-tête de l'expression. Travailler ensuite les éléments (parties) de l'expression. Si un élément n'est pas une expression atomique: Procéder par parties etc.. Ce procédé est appliqué récursivement.

9.9.2. Ausnahmefall

■ Exception

Wenn ein Element oder Ausdruck das Attribut HoldFirst, HoldAll, HoldRest hat oder die Funktion Evaluate aufruft, wird die Voreinstellung durchbrochen. Erkunde, wo solche Attribute vorkommen:

■ Quand un élément en une expression a l'attribut HoldFirst, HoldAll, HoldRest ou appellent la fonction Evaluate, le règlement préalable est interrompu. Cherche où se trouvent de tels attributs:

In[139]:=

??Sin

Sin[z] gives the sine of z. Mehr...

Attributes[Sin] = {Listable, NumericFunction, Protected}

In[140]:=

??Plot

Plot[f, {x, xmin, xmax}] generates a plot of f as a function of x from xmin to xmax. Plot[{f1, f2, ... }, {x, xmin, xmax}] plots several functions fi. Mehr...

Attributes[Plot] = {HoldAll, Protected}

Options[Plot] = {AspectRatio $\rightarrow \frac{1}{\text{GoldenRatio}}$, Axes $\rightarrow$ Automatic, AxesLabel $\rightarrow$ None, AxesOrigin $\rightarrow$ Automatic, AxesStyle $\rightarrow$ Automatic, Background $\rightarrow$ Automatic, ColorOutput $\rightarrow$ Automatic, Compiled $\rightarrow$ True, DefaultColor $\rightarrow$ Automatic, DefaultFont $\rightarrow$ \$DefaultFont, DisplayFunction $\rightarrow$ \$DisplayFunction, Epilog $\rightarrow$ {}, FormatType $\rightarrow$ \$FormatType, Frame $\rightarrow$ False, FrameLabel $\rightarrow$ None, FrameStyle $\rightarrow$ Automatic, FrameTicks $\rightarrow$ Automatic, GridLines $\rightarrow$ None, ImageSize $\rightarrow$ Automatic, MaxBend $\rightarrow$ 10., PlotDivision $\rightarrow$ 30., PlotLabel $\rightarrow$ None, PlotPoints $\rightarrow$ 25, PlotRange $\rightarrow$ Automatic, PlotRegion $\rightarrow$ Automatic, PlotStyle $\rightarrow$ Automatic, Prolog $\rightarrow$ {}, RotateLabel $\rightarrow$ True, TextStyle $\rightarrow$ \$TextStyle, Ticks $\rightarrow$ Automatic}

In[141]:=

??Table

Table[expr, {imax}] generates a list of imax copies of expr. Table[expr, {i, imax}] generates a list of the values of expr when i runs from 1 to imax. Table[expr, {i, imin, imax}] starts with i = imin. Table[expr, {i, imin, imax, di}] uses steps di. Table[expr, {i, imin, imax}, {j, jmin, jmax}, ...] gives a nested list. The list associated with i is outermost. Mehr...

Attributes[Table] = {HoldAll, Protected}

In[142]:=

??Attributes

Attributes[symbol] gives the list of attributes for a symbol. Mehr...

Attributes[Attributes] = {HoldAll, Listable, Protected}

In[143]:=

??Evaluate

Evaluate[expr] causes expr to be evaluated even if it appears as the argument of a function whose attributes specify that it should be held unevaluated. Mehr...

Attributes[Evaluate] = {Protected}

Vergleiche die Resultate bei Abänderung der Abarbeitungsreihenfolge und erkläre den Unterschied der Resultate:

■ Compare les résultats lors des changements de l'ordre de travail et explique la différence des résultats:

In[144]:=

Table[Random[], {5}]

Out[144]=

{0.258329, 0.518034, 0.790079, 0.821881, 0.860525}

In[145]:=

Table[Evaluate[Random[]], {5}]

Out[145]=

{0.581881, 0.581881, 0.581881, 0.581881, 0.581881}

Wieso kommt das so, wie es kommt?

■ Pourquoi cela se déroule ainsi?

9.10. Diskrete Funktionen

■ Fonctions discrètes

Ein Beispiel:

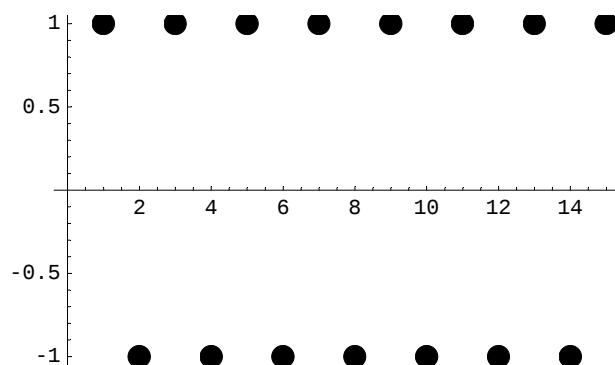
■ Un exemple:

```
In[146]:=
Clear[ungerade, t];
ungerade[x_?OddQ] := (ungerade[x] = 1);
ungerade[x_?EvenQ] := (ungerade[x] = -1);

In[149]:=
t = Table[ungerade[x], {x, 1, 15}]

Out[149]=
{1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1}
```

```
In[150]:=
ListPlot[t, PlotStyle -> PointSize[0.04]];
```



```
In[151]:=
?ListPlot

ListPlot[{y1, y2, ...}] plots points corresponding to a list of values. The x
coordinates are by default taken to be 1, 2, ... . ListPlot[{x1, y1}, {x2,
y2}, ...] plots a list of points with specified x and y coordinates. Mehr...
```

"Putzmaschine" einsetzen

■ Employer la "machine de nettoyage"

```
In[152]:=
(* Old Form: Remove["Global`*"] *)
```

```
In[153]:=
Remove["Global`*"]
```