Kurs ▮ Cours

10. Lokale Variablen und prozedurales Programmieren
▮ Variables locales et programmation procédurale

Die Gliederung dieses Kurses folgt in groben Zügen dem Buch von Nancy Blachman: A Practical Approach....
Hinweis: Kapitel 10 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 10.
Testé avec *Mathematica* version 5.2+WIN

10.1. Globale Variablen
▮ Variables globales

Mathematica stört es nicht, dass im folgenden Beispiel zweimal dieselbe globale Variable (hier "i") in verschiedenen Bedeutungen gebraucht wird:
▮ Cela ne dérange pas *Mathematica* que dans l'exemple suivant on emploie deux fois la même variable globale (ici "i") avec deux significations différentes:

```
In[1]:= f[n_]:=Table[g[i],  {i,n}];
        g[m_]:=Table[Log[i],{i,m}];

In[3]:= f[3]

Out[3]= {{0}, {0, Log[2]}, {0, Log[2], Log[3]}}

In[4]:= ??i

        Global`i
```

Zum Vergleich: ▮ Pour comparer:

```
In[5]:= f[n_]:=Table[g[i],  {i,n}];
        g[m_]:=Table[Log[j],{j,m}];
```

```
In[7]:= f[3]

Out[7]= {{0}, {0, Log[2]}, {0, Log[2], Log[3]}}
```

10.2. Lokale Variablen im Unterschied zu globalen Variablen
▮ Variables locales à la différence de variables globales

10.2.1. Lokale Variablen in einer "Block"-Prozedur
▮ Variables locales dans une procédure "Block"

```
In[8]:= ??Block

        Block[{x, y, ... }, expr] specifies that expr is
           to be evaluated with local values for the symbols x, y, ... . Block[
           {x = x0, ... }, expr] defines initial local values for x, ... . Mehr...

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

Betrachte das folgende Beispiel:
▮ Considère l'exemple suivant:

```
In[9]:= Clear[f,g];
        f[n_] := Block[{i}, Table[  g[i], {i, n}]];
        g[m_] := Block[{i}, Table[Log[i], {i, m}]]
```

Ausprobieren: ▮ Essayer:

```
In[12]:= i

Out[12]= i

In[13]:= ?f

        Global`f

        f[n_] := Block[{i}, Table[g[i], {i, n}]]

In[14]:= f[n]

        Table::iterb : Iterator {i, n} does not have appropriate bounds. Mehr…

        Table::iterb : Iterator {i, n} does not have appropriate bounds. Mehr…

Out[14]= Table[g[i], {i, n}]

In[15]:= f[3]

Out[15]= {{0}, {0, Log[2]}, {0, Log[2], Log[3]}}

In[16]:= f[5]

Out[16]= {{0}, {0, Log[2]}, {0, Log[2], Log[3]},
          {0, Log[2], Log[3], Log[4]}, {0, Log[2], Log[3], Log[4], Log[5]}}
```

In[17]:=
```
i
```

Out[17]=
```
i
```

Was ist passiert? (Welcher Wert ist in "i" gespeichert?) Betrachte:

■ Que s'est-il passé? (Quelle valeur est mémorisée dans "i"?) Considère:

In[18]:=
```
Clear[f,g,i];
Block[{i}, i]
```

Out[19]=
```
i
```

10.2.2. Lokale Variablen in einer "Module"-Prozedur
■ Variables locales dans une procédure "Module"

In[20]:=
```
??Module
```

```
Module[{x, y, ... }, expr] specifies that occurrences
   of the symbols x, y, ...  in expr should be treated as local. Module[
   {x = x0, ... }, expr] defines initial values for x, ... . Mehr...

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

Vergleiche mit dem letzten Beispiel:

■ Compare au dernier exemple:

In[21]:=
```
Clear[f];
Module[{i}, i]
```

Out[22]=
```
i$36
```

In[23]:=
```
Clear[i];
Module[{i}, i]
```

Out[24]=
```
i$37
```

Was bedeutet wohl die ausgegebene Zahl?

■ Que signifie le nombre sorti?

Betrachte das folgende Beispiel:

■ Considère l'exemple suivant:

In[25]:=
```
Clear[f,g];
f[n_] := Module[{i}, Table[  g[i], {i, n}]];
g[m_] := Module[{i}, Table[Log[i], {i, m}]]
```

Ausprobieren: ■ Essayer:

In[28]:=
```
?f
```

```
Global`f

f[n_] := Module[{i}, Table[g[i], {i, n}]]
```

In[29]:= **f[n]**

```
Table::iterb : Iterator {i$42, n} does not have appropriate bounds. Mehr…
```

Out[29]= `Table[g[i$42], {i$42, n}]`

In[30]:= **f[3]**

Out[30]= `{{0}, {0, Log[2]}, {0, Log[2], Log[3]}}`

In[31]:= **f[5]**

Out[31]=
```
{{0}, {0, Log[2]}, {0, Log[2], Log[3]},
 {0, Log[2], Log[3], Log[4]}, {0, Log[2], Log[3], Log[4], Log[5]}}
```

In[32]:= **i**

Out[32]= `i`

Was ist passiert? (Welcher Wert ist in "i" gespeichert?)

- Que s'est-il passé? (Quelle valeur est mémorisée dans "i"?)

10.2.3. Lokale Variablen mit "With"
Variables locales avec "With"

Damit lassen sich Ausdrücke manipulieren:

- Par cela on peut manipuler des expressions:

In[33]:= **?With**

```
With[{x = x0, y = y0, ... }, expr] specifies that in expr occurrences
   of the symbols x, y, ...  should be replaced by x0, y0, ... . Mehr...
```

In[34]:= **With[{x=2, y=a, z=b^3}, y Log[x^z]]**

Out[34]= $a \operatorname{Log}\left[2^{b^3}\right]$

In[35]:=
```
Clear[x,y,z,r,s];
Solve[{x+2==r-s,s+1==y+2z,2r+z==-4x},{x,r}]
```

Out[36]= $\left\{\left\{x \to \frac{1}{6}\,(-4-2\,s-z),\ r \to \frac{1}{6}\,(8+4\,s-z)\right\}\right\}$

In[37]:=
```
With[{z=4, s=Sin[z^2]},
Solve[{x+2==r-s,s+1==y+2z,2r+z==-4x},{x,r}]]
```

Out[37]= $\left\{\left\{x \to \frac{1}{3}\,(-4-\operatorname{Sin}[z^2]),\ r \to \frac{2}{3}\,(1+\operatorname{Sin}[z^2])\right\}\right\}$

10.3. Prozedurale Programmierung
▮ Programmation procédurale

Mathematica erlaubt prozedurale Programmierung, d.h. Befehlssequenzen mit Wiederholungen und Verzweigungen. Studiere die folgenden Befehle:

▮ *Mathematica* permet la programmation procédurale, c'est-à-dire des séquences d'ordres avec des répétitions et des ramifications. Etudier les ordres suivants:

```
In[38]:= ??Do
```

```
Do[expr, {imax}] evaluates expr imax times. Do[expr, {i, imax}] evaluates expr
   with the variable i successively taking on the values 1 through imax (in steps
   of 1). Do[expr, {i, imin, imax}] starts with i = imin. Do[expr, {i, imin,
   imax, di}] uses steps di. Do[expr, {i, imin, imax}, {j, jmin, jmax}, ... ]
   evaluates expr looping over different values of j, etc. for each i. Mehr...

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

```
In[39]:= ?For
```

```
For[start, test, incr, body] executes start, then
   repeatedly evaluates body and incr until test fails to give True. Mehr...
```

```
In[40]:= ?While
```

```
While[test, body] evaluates test, then
   body, repetitively, until test first fails to give True. Mehr...
```

```
In[41]:= ?If
```

```
If[condition, t, f] gives t if condition evaluates to True, and f if it evaluates to False.
   If[condition, t, f, u] gives u if condition evaluates to neither True nor False. Mehr...
```

```
In[42]:= ?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[43]:= ?Switch
```

```
Switch[expr, form1, value1, form2, value2, ... ] evaluates
   expr, then compares it with each of the formi in turn, evaluating and
   returning the valuei corresponding to the first match found. Mehr...
```

10.3.1. Arithmetische Operationen auf bestehenden Variablen wie Inkrementierung u.s.w.
▮ Opérations arithmétiques sur des variables existantes comme "increment" etc.

Studiere die folgenden Beispiele:

▮ Etudie les exemples suivants:

```
In[44]:= ??++
```

```
x++ increases the value of x by 1, returning the old value of x. Mehr...

Attributes[Increment] = {HoldFirst, Protected, ReadProtected}
```

In[45]:= **?+=**

x += dx adds dx to x and returns the new value of x. Mehr...

In[46]:= **?-=**

x -= dx subtracts dx from x and returns the new value of x. Mehr...

In[47]:= **?*=**

x *= c multiplies x by c and returns the new value of x. Mehr...

In[48]:= **?/=**

x /= c divides x by c and returns the new value of x. Mehr...

In[49]:= **?DivideBy**

x /= c divides x by c and returns the new value of x. Mehr...

In[50]:= **i=1**

Out[50]= 1

In[51]:= **++i**

Out[51]= 2

In[52]:= **i**

Out[52]= 2

In[53]:= **i++**

Out[53]= 2

In[54]:= **i**

Out[54]= 3

In[55]:= **i**

Out[55]= 3

In[56]:= **i +=6**

Out[56]= 9

In[57]:= **i -=5**

Out[57]= 4

In[58]:= **i *=7**

Out[58]= 28

In[59]:= **i /=4**

Out[59]= 7

10.3.2. Iterative Konstruktionen
▮ Constructions itératives

Studiere die folgenden Beispiele:
▮ Etudie les exemples suivants:

```
In[60]:= Do[Print[3 n^2], {n,7}]
```

```
3
12
27
48
75
108
147
```

```
In[61]:= For[n=1, n<=7, ++n, Print[3 n^2]]
```

```
3
12
27
48
75
108
147
```

```
In[62]:= n=1;
         While[n<=7, (Print[3 n^2]; n++)]
```

```
3
12
27
48
75
108
147
```

10.3.3. Logik
▮ Logique

Studiere die folgenden Befehle:
▮ Etudie les ordres suivants:

In[64]:= **?&&**

e1 && e2 && ... is the logical AND function. It evaluates its arguments in order, giving
False immediately if any of them are False, and True if they are all True. Mehr...

In[65]:= **?And**

e1 && e2 && ... is the logical AND function. It evaluates its arguments in order, giving
False immediately if any of them are False, and True if they are all True. Mehr...

In[66]:= **?||**

e1 || e2 || ... is the logical OR function. It evaluates its arguments in order, giving
True immediately if any of them are True, and False if they are all False. Mehr...

In[67]:= **?Or**

e1 || e2 || ... is the logical OR function. It evaluates its arguments in order, giving
True immediately if any of them are True, and False if they are all False. Mehr...

In[68]:= **?!**

Information::notfound : Symbol ! not found. Mehr…

In[69]:= **?!!**

n!! gives the double factorial of n. Mehr...

In[70]:= **?Not**

!expr is the logical NOT function. It gives False if expr is True, and True if it is False.
Mehr…

In[71]:= **?Xor**

Xor[e1, e2, ...] is the logical XOR (exclusive OR) function. It gives
True if an odd number of the ei are True, and the rest are False. It gives
False if an even number of the ei are True, and the rest are False. Mehr...

Studiere die folgenden Beispiele:

■ Etudie les exemples suivants:

In[72]:= **(1 < 2) || (4 > 5/0)**

Out[72]= True

In[73]:= **(4 > 5/0) || (1 < 2)**

Power::infy : Infinite expression $\frac{1}{0}$ encountered. Mehr…

Greater::nord : Invalid comparison with ComplexInfinity attempted. Mehr…

Out[73]= True

Von wo her wird also geklammert?

■ D'où viennent les parenthèses?

10.3.4. Wahrheitstabellen
▮ Tableaux de verité

Ein Beispiel: Die folgende Aussageform soll untersucht werden:

▮ Un exemple: La forme propositionelle suivante soit examinée:

```
In[74]:= Clear[f];
         f[x_:0, y_:0, z_:0, w_:0, u_:0, v_:0]:=
                 !(!x || (x && y)) || (x || ! y) && w;
         AppendTo[Attributes[f], Listable]

Out[76]= {Listable}

In[77]:= ??f

        Global`f

         Attributes[f] = {Listable}

         f[x_ : 0, y_ : 0, z_ : 0, w_ : 0, u_ : 0, v_ : 0] := ! (! x || (x && y)) || ((x || ! y) && w)
```

Die nachstehende Funktion definiert den "Input" in die Wahrheitstabelle:

▮ La fonction ci-dessous définit l'"Input" dans le tableau de vérité:

```
In[78]:= Remove[tabteil,tabelle];
         tabteil[n_, k_]:= Permutations[
                         Table[Ceiling[Floor[n/i]/k],
                                 {i,1,k}]];
         tabelle = {};
         t[k_]:=(Do[AppendTo[tabelle,tabteil[n,k]],
                                 {n,0,k}];
                 tabelle  = Sort[Flatten[tabelle,1]];
                 (tabelleTF=tabelle /. {0->False,
                                        1->True});
                 Print[MatrixForm[tabelle]];
                 MatrixForm[tabelleTF]);
         t[3]
```

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

```
Out[82]//MatrixForm=
```

$$\begin{pmatrix} \text{False} & \text{False} & \text{False} \\ \text{False} & \text{False} & \text{True} \\ \text{False} & \text{True} & \text{False} \\ \text{False} & \text{True} & \text{True} \\ \text{True} & \text{False} & \text{False} \\ \text{True} & \text{False} & \text{True} \\ \text{True} & \text{True} & \text{False} \\ \text{True} & \text{True} & \text{True} \end{pmatrix}$$

```
In[83]:= Remove[tabteil,tabelle];
         tabteil[n_, k_]:= Permutations[
                         Table[Ceiling[Floor[n/i]/k],
                                        {i,1,k}]];
         tabelle = {};
         t[k_]:=(Do[AppendTo[tabelle,tabteil[n,k]],
                                        {n,0,k}];
                 tabelle  = Sort[Flatten[tabelle,1]];
                 (tabelleTF=tabelle /. {0->False,
                                        1->True});
                 Print[MatrixForm[tabelle]];
                 MatrixForm[tabelleTF]);
         t[4]
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Out[87]//MatrixForm=

$$\begin{pmatrix} \text{False} & \text{False} & \text{False} & \text{False} \\ \text{False} & \text{False} & \text{False} & \text{True} \\ \text{False} & \text{False} & \text{True} & \text{False} \\ \text{False} & \text{False} & \text{True} & \text{True} \\ \text{False} & \text{True} & \text{False} & \text{False} \\ \text{False} & \text{True} & \text{False} & \text{True} \\ \text{False} & \text{True} & \text{True} & \text{False} \\ \text{False} & \text{True} & \text{True} & \text{True} \\ \text{True} & \text{False} & \text{False} & \text{False} \\ \text{True} & \text{False} & \text{False} & \text{True} \\ \text{True} & \text{False} & \text{True} & \text{False} \\ \text{True} & \text{False} & \text{True} & \text{True} \\ \text{True} & \text{True} & \text{False} & \text{False} \\ \text{True} & \text{True} & \text{False} & \text{True} \\ \text{True} & \text{True} & \text{True} & \text{False} \\ \text{True} & \text{True} & \text{True} & \text{True} \end{pmatrix}$$

Und hier der "Output" in der entsprechenden Reihenfolge:

■ Et voici l'"Output" dans l'ordre correspondant:

```
In[88]:= ??Map
```

```
Map[f, expr] or f /@ expr applies f to each element on the first level in expr. Map[
   f, expr, levelspec] applies f to parts of expr specified by levelspec. Mehr...

Attributes[Map] = {Protected}

Options[Map] = {Heads → False}
```

In[89]:=
```
f @@ Transpose[tabelleTF]
```

Out[89]=
```
{False, True, False, True, False, False, False,
 False, True, True, True, True, False, True, False, True}
```

In[90]:=
```
MatrixForm[Transpose[
    Join[Transpose[tabelleTF],
    {Table["|",{i,1,
        Length[f @@ Transpose[tabelleTF]]}]},
    {f @@ Transpose[tabelleTF]}]]]
```

Out[90]//MatrixForm=

False	False	False	False	|	False
False	False	False	True	|	True
False	False	True	False	|	False
False	False	True	True	|	True
False	True	False	False	|	False
False	True	False	True	|	False
False	True	True	False	|	False
False	True	True	True	|	False
True	False	False	False	|	True
True	False	False	True	|	True
True	False	True	False	|	True
True	False	True	True	|	True
True	True	False	False	|	False
True	True	False	True	|	True
True	True	True	False	|	False
True	True	True	True	|	True

```
In[91]:= Remove[tabteil,tabelle];
         tabteil[n_, k_]:= Permutations[
                         Table[Ceiling[Floor[n/i]/k],
                                              {i,1,k}]];
         tabelle = {};
         t[k_]:=(Do[AppendTo[tabelle,tabteil[n,k]],
                                              {n,0,k}];
                 tabelle  = Sort[Flatten[tabelle,1]];
                 (tabelleTF=tabelle /. {0->False,
                                              1->True});
                 MatrixForm[Transpose[
                 Join[Transpose[tabelleTF],
                 {Table["|",{i,1,
                     Length[f @@ Transpose[tabelleTF]]}]},
                 {f @@ Transpose[tabelleTF]}]]]);
         (*Anwendung*)
         t[5]
```

Out[96]//MatrixForm=

False	False	False	False	False	|	False
False	False	False	False	True	|	False
False	False	False	True	False	|	True
False	False	False	True	True	|	True
False	False	True	False	False	|	False
False	False	True	False	True	|	False
False	False	True	True	False	|	True
False	False	True	True	True	|	True
False	True	False	False	False	|	False
False	True	False	False	True	|	False
False	True	False	True	False	|	False
False	True	False	True	True	|	False
False	True	True	False	False	|	False
False	True	True	False	True	|	False
False	True	True	True	False	|	False
False	True	True	True	True	|	False
True	False	False	False	False	|	True
True	False	False	False	True	|	True
True	False	False	True	False	|	True
True	False	False	True	True	|	True
True	False	True	False	False	|	True
True	False	True	False	True	|	True
True	False	True	True	False	|	True
True	False	True	True	True	|	True
True	True	False	False	False	|	False
True	True	False	False	True	|	False
True	True	False	True	False	|	True
True	True	False	True	True	|	True
True	True	True	False	False	|	False
True	True	True	False	True	|	False
True	True	True	True	False	|	True
True	True	True	True	True	|	True

10.3.5. Verzweigungen
▮ Ramification

Probiere aus: ▮ Essaie:

In[97]:= **?If**

If[condition, t, f] gives t if condition evaluates to True, and f if it evaluates to False.
If[condition, t, f, u] gives u if condition evaluates to neither True nor False. Mehr...

In[98]:= **?PrintForm**

System`PrintForm

Attributes[PrintForm] = {Protected}

In[99]:= **PrintForm[Print"gagag"]**

Out[99]= HorizontalForm[{11, 1, 0, 0, {10, 4, 10, 2, 6}},
{Times, 400, None}, , , gagag, , Print,]

In[100]:=
?HorizontalForm

HorizontalForm is an internal symbol used for formatting and printing.

In[101]:=
?Write

Write[channel, expr1, expr2, ...] writes the expressions expri in
sequence, followed by a newline, to the specified output channel. Mehr...

In[102]:=
?WriteString

WriteString[channel, expr1, expr2, ...] converts the expri to strings,
and then writes them in sequence to the specified output channel. Mehr...

In[103]:=
?SequenceForm

SequenceForm[expr1, expr2, ...] prints as the
textual concatenation of the printed forms of the expri. Mehr...

In[104]:=

```
Clear[t];
Do[(t=Table[(If[Random[]>0.5,
    SequenceForm["omumo"],SequenceForm["wvovw"],
    SequenceForm["bubu"]]),{i,7}];
Print[t[[1]],t[[2]],t[[3]],t[[4]],t[[5]],
        t[[6]],t[[7]]]),{k,25}]
```

```
wvovwomumowvovwomumoomumowvovwwvovw

wvovwwvovwomumowvovwomumoomumoomumo

wvovwwvovwomumowvovwwvovwwvovwwvovw

omumoomumowvovwomumowvovwwvovwwvovw

wvovwwvovwwvovwomumowvovwwvovwomumo

wvovwwvovwwvovwomumowvovwomumowvovw

wvovwwvovwomumowvovwomumoomumoomumo

wvovwomumoomumowvovwwvovwomumoomumo

wvovwomumoomumowvovwomumowvovwwvovw

wvovwwvovwwvovwomumowvovwomumoomumo

wvovwwvovwomumoomumowvovwomumoomumo

omumoomumoomumowvovwwvovwomumoomumo

wvovwomumowvovwwvovwwvovwomumoomumo

wvovwomumoomumowvovwomumoomumowvovw

wvovwwvovwwvovwwvovwwvovwwvovwomumo

wvovwomumowvovwomumoomumowvovwwvovw

wvovwomumoomumowvovwwvovwwvovwwvovw

wvovwomumowvovwwvovwwvovwwvovwomumo

omumowvovwwvovwwvovwwvovwomumoomumo

omumoomumoomumoomumoomumowvovwwvovw

wvovwwvovwwvovwwvovwomumowvovwwvovw

omumowvovwomumoomumoomumoomumoomumo

omumowvovwwvovwwvovwomumoomumoomumo

omumowvovwwvovwomumowvovwomumoomumo

omumoomumoomumowvovwomumoomumoomumo
```

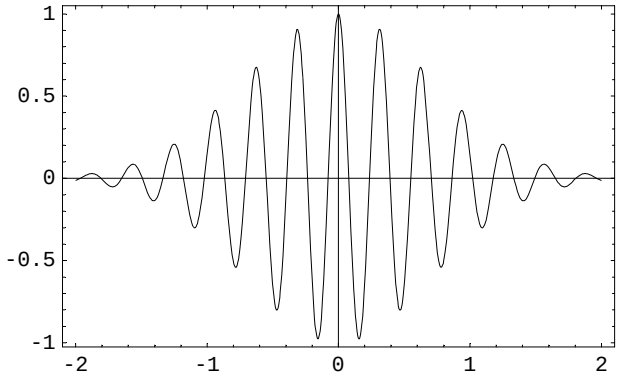
Verzweigung je nach *Mathematica*-Version:

■ Ramification selon la version de *Mathematica*:

In[106]:=

```
If[$VersionNumber < 2.0,
    Plot[E^(-x^2) Cos[20x],{x,-2,2},Framed->True],
    Plot[E^(-x^2) Cos[20x],{x,-2,2},Frame ->True]];
```

```
General::spell1 :
 Possible spelling error: new symbol name "Framed" is similar to existing symbol "Frame". Mehr…
```



Mit Switch: ■ Avec Switch:

In[107]:=

```
x=Random[Integer,{1,5}];
Print[x];
Switch[x^2,1,x,4,2x,9,3x,16,4x,25,-5x]
```

```
1
```

Out[109]=

```
1
```

In[110]:=

```
x=Random[Integer,{1,5}];
Print[x];
Switch[x^2,1,x,4,2x,9,3x,16,4x,25,-5x]
```

```
1
```

Out[112]=

```
1
```

In[113]:=

```
x=Random[Integer,{1,5}];
Print[x];
Switch[x^2,1,x,4,2x,9,3x,16,4x,25,-5x]
```

```
4
```

Out[115]=

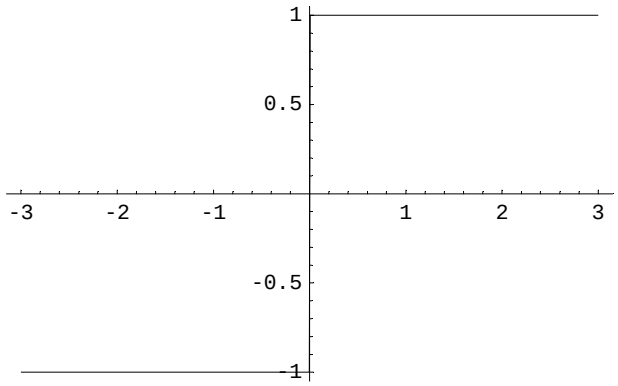
```
16
```

Mit Which: ■ Avec Which:

In[116]:=

```
signum[x_]:=Which[x<0.,-1,x==0.,0,x>0.,1]
```

In[117]:=
```
Plot[signum[x],{x,-3,3}];
```



10.3.6: Spaghetti-Code (resp. die "Chaos-Erzeugung"...)
▮ Code spaghetti (résp. la "génération du chaos"...)

Davon ist abzuraten!
▮ On le déconseille!

In[118]:=
```
?Goto
```

```
Goto[tag] scans for Label[tag], and transfers control to that point. Mehr...
```

In[119]:=
```
?Label
```

```
Label[tag] represents a point in a compound
   expression to which control can be transferred using Goto. Mehr...
```

In[120]:=
```
?Throw
```

```
Throw[value] stops evaluation and returns value as the
   value of the nearest enclosing Catch. Throw[value, tag] is caught only
   by Catch[expr, form] where form is a pattern that matches tag. Mehr...
```

In[121]:=
```
?Catch
```

```
Catch[expr] returns the argument of the first Throw generated in the evaluation
   of expr. Catch[expr, form] returns value from the first Throw[value, tag] for
   which form matches tag. Catch[expr, form, f] returns f[value, tag]. Mehr...
```

10.3.7: Beispiel aus der Statistik
▮ Exemple pris de la statistique

Der Median: ▮ La médiane:

In[122]:=
```
median[liste_List]:=
    Block[{
            sl,
            len
          },
          len = Length[liste];
          sl = Sort[liste];
          If[
             OddQ[len],
             sl[[(len+1)/2]],
             (sl[[len/2]]+sl[[len/2+1]])/2
          ]
    ]
```

```
General::spell1 :
 Possible spelling error: new symbol name "median" is similar to existing symbol "Median". Mehr…
```

In[123]:=
```
median[{53,64,78,24,63,78}] // N
```

Out[123]=
```
63.5
```

In[124]:=
```
median[{76, 56, 23, 78, 34}]
```

Out[124]=
```
56
```

In[125]:=
```
median[{0,1,2,3,4,5,6}]
```

Out[125]=
```
3
```

In[126]:=
```
median[{1,2,3,4,5,6}]  // N
```

Out[126]=
```
3.5
```

In[127]:=
```
median[{1,2,3,4,5,6,7}]
```

Out[127]=
```
4
```

Was ist der Median?

- Qu'est-ce la médiane?

"Putzmaschine" einsetzen
Employer la "machine de nettoyage"

In[128]:=
```
(* Old Form: Remove["Global`@*"] *)
```

In[129]:=
```
Remove["Global`*"]
```