# Kurs ▮ Cours

## 6. Manipulation von Listen
## ▮ Manipulation de listes

Die Gliederung dieses Kurses folgt in groben Zügen dem Buch von Nancy Blachman: A Practical Approach....
Hinweis: Kapitel 6 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 6.
Testé avec *Mathematica* version 5.2+WIN



---

## 6.0. Einleitung
## ▮ Introduction

Listen - was sind Listen - was ist das?
▮ Des listes - qu'est - ce?

```
In[1]:= ??List

        {e1, e2, ... } is a list of elements. Mehr...

        Attributes[List] = {Locked, Protected}
```

Eingabe einer Liste:
▮ Entrer une liste:

```
In[2]:= list0 = {5, 4.78, a, 8x + y, Sin[x], D,
        Integer, x->E}

Out[2]= {5, 4.78, a, 8 x + y, Sin[x], D, Integer, x → e}
```

Oft gibt *Mathematica* auch Listen aus. Beispiel:
▮ *Mathematica* sort souvent des listes. Exemple:

*In[3]:=* **Solve[{2x + 5y == 19, 3x + 7y == 27}]**

*Out[3]=* {{x → 2, y → 3}}

Hier ist eine Liste mit einer Liste ausgegeben worden.
- Ici on a sorti une liste qui contient une liste.

Ebenso z.B. bei "Options":
- Egalement p.ex. avec "Options":

*In[4]:=* **Options[NIntegrate]**

*Out[4]=* {AccuracyGoal → ∞, Compiled → True, EvaluationMonitor → None, GaussPoints → Automatic, MaxPoints → Automatic, MaxRecursion → 6, Method → Automatic, MinRecursion → 0, PrecisionGoal → Automatic, SingularityDepth → 4, WorkingPrecision → MachinePrecision}

Hier ist eine Liste ausgegeben worden.
- Ici on a sorti une liste.

---

# 6.1. Erzeugung von Listen
# ▮ Générer des listes

## 6.1.1. Befehle
## ▮ Ordres

Zur Erzeugung von Listen stehen verschiedene Befehle zur Verfügung:
- Pour créer des listes on a plusieurs ordres à disposition:

*In[5]:=* **??Range**

```
Range[imax] generates the list {1, 2, ... , imax}. Range[imin, imax] generates
   the list {imin, ... , imax}. Range[imin, imax, di] uses step di. Mehr...

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

*In[6]:=* **??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[7]:=* **??Array**

```
Array[f, n] generates a list of length n, with elements f[i]. Array[
   f, {n1, n2, ... }] generates an n1 by n2 by ... array of nested lists,
   with elements f[i1, i2, ... ]. Array[f, {n1, n2, ... }, {r1, r2, ... }]
   generates a list using the index origins ri (default 1). Array[f, dims,
   origin, h] uses head h, rather than List, for each level of the array. Mehr...

Attributes[Array] = {Protected}
```

### 6.1.2. Erzeugung einer Liste mit "Range"
### ▮ Créer une liste avec "Range"

Beispiele: ▮ Exemples:

```
In[8]:= Range[5]
Out[8]= {1, 2, 3, 4, 5}

In[9]:= Range[7.896]
Out[9]= {1, 2, 3, 4, 5, 6, 7}

In[10]:= Range[6,12]
Out[10]= {6, 7, 8, 9, 10, 11, 12}

In[11]:= Range[3.4, 8.7]
Out[11]= {3.4, 4.4, 5.4, 6.4, 7.4, 8.4}

In[12]:= Range[3.4, 8.7, 1.2]
Out[12]= {3.4, 4.6, 5.8, 7., 8.2}
```

### 6.1.3. Erzeugung einer Liste mit "Table"
### ▮ Générer des listes avec "Table"

Beispiele: ▮ Exemples:

```
In[13]:= Table[hoi,{6}]
Out[13]= {hoi, hoi, hoi, hoi, hoi, hoi}

In[14]:= Table[i!,{i, 6}]
Out[14]= {1, 2, 6, 24, 120, 720}

In[15]:= Table[3i,{i, 6, 14}]
Out[15]= {18, 21, 24, 27, 30, 33, 36, 39, 42}

In[16]:= Table[i,{i, 4, 40, 5}]
Out[16]= {4, 9, 14, 19, 24, 29, 34, 39}

In[17]:= Table[{i, i^2},{i, 5}]
Out[17]= {{1, 1}, {2, 4}, {3, 9}, {4, 16}, {5, 25}}

In[18]:= Table[a[i] + b[j], {i, 1, 3}, {j, 4, 6}]
Out[18]= {{a[1] + b[4], a[1] + b[5], a[1] + b[6]},
          {a[2] + b[4], a[2] + b[5], a[2] + b[6]}, {a[3] + b[4], a[3] + b[5], a[3] + b[6]}}
```

Das ist eine Liste von Listen, d.h. eine Matrix.
Zur besseren Darstellung verwenden wir "MatrixForm":

■ C'est une liste de listes, c'est-à-dire une matrice.
Pour plus de clarté nous utilisons "MatrixForm"

```
In[19]:= MatrixForm[%]
```

```
Out[19]//MatrixForm=
        ⎛a[1] + b[4]  a[1] + b[5]  a[1] + b[6]⎞
        ⎜a[2] + b[4]  a[2] + b[5]  a[2] + b[6]⎟
        ⎝a[3] + b[4]  a[3] + b[5]  a[3] + b[6]⎠
```

Ein Iterator kann auch von einem andern abhängen:
■ Un itérateur peut aussi dépendre d'un autre:

```
In[20]:= Table[a[i] + b[j], {i, 1, 3}, {j, i, 6}]
```

```
Out[20]= {{a[1] + b[1], a[1] + b[2], a[1] + b[3], a[1] + b[4], a[1] + b[5], a[1] + b[6]},
          {a[2] + b[2], a[2] + b[3], a[2] + b[4], a[2] + b[5], a[2] + b[6]},
          {a[3] + b[3], a[3] + b[4], a[3] + b[5], a[3] + b[6]}}
```

```
In[21]:= MatrixForm[%]
```

```
Out[21]//MatrixForm=
        ⎛{a[1] + b[1], a[1] + b[2], a[1] + b[3], a[1] + b[4], a[1] + b[5], a[1] + b[6]}⎞
        ⎜{a[2] + b[2], a[2] + b[3], a[2] + b[4], a[2] + b[5], a[2] + b[6]}              ⎟
        ⎝{a[3] + b[3], a[3] + b[4], a[3] + b[5], a[3] + b[6]}                          ⎠
```

### 6.1.4. "Array"
#### ■ "Array"

Mit "Array" kann man eine symbolische Matrix erzeugen:
■ Avec "Array" on peut créer une matrice symbolique:

```
In[22]:= Clear[a]
```

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

```
Out[23]= {{a[1, 1], a[1, 2], a[1, 3]}, {a[2, 1], a[2, 2], a[2, 3]}}
```

```
In[24]:= MatrixForm[%]
```

```
Out[24]//MatrixForm=
        ⎛a[1, 1]  a[1, 2]  a[1, 3]⎞
        ⎝a[2, 1]  a[2, 2]  a[2, 3]⎠
```

```
In[25]:= a = 17; Array[a,{2, 3}] // MatrixForm
```

```
Out[25]//MatrixForm=
        ⎛17[1, 1]  17[1, 2]  17[1, 3]⎞
        ⎝17[2, 1]  17[2, 2]  17[2, 3]⎠
```

# 6.2. Umordnen von Listen
## ▮ Ranger autrement des listes

### 6.2.1. Befehle
#### ▮ Ordres

*In[26]:=* **??Sort**

```
Sort[list] sorts the elements of list into canonical
   order. Sort[list, p] sorts using the ordering function p.
```
Mehr...

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

*In[27]:=* **??Reverse**

```
Reverse[expr] reverses the order of the elements in expr.
```
Mehr...

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

*In[28]:=* **??RotateLeft**

```
RotateLeft[expr, n] cycles the elements in expr n positions to the left.
   RotateLeft[expr] cycles one position to the left. RotateLeft[expr, {n1,
   n2, ... }] cycles elements at successive levels ni positions to the left.
```
Mehr...

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

*In[29]:=* **??RotateRight**

```
RotateRight[expr, n] cycles the elements in expr n positions to the right.
   RotateRight[expr] cycles one position to the right. RotateRight[expr, {n1,
   n2, ... }] cycles elements at successive levels ni positions to the right.
```
Mehr...

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

*In[30]:=* **??Permutations**

```
Permutations[list] generates a list of all possible permutations of the elements in list.
```
Mehr...

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

*In[31]:=* **??Drop**

```
Drop[list, n] gives list with its first n elements dropped. Drop[list,
   -n] gives list with its last n elements dropped. Drop[list, {n}] gives list
   with its nth element dropped. Drop[list, {m, n}] gives list with elements m
   through n dropped. Drop[list, {m, n, s}] gives list with elements m through
   n in steps of s dropped. Drop[list, seq1, seq2, ... ] gives a nested list in
   which elements specified by seqi have been dropped at level i in list.
```
Mehr...

```
Attributes[Drop] = {NHoldRest, Protected}
```

*In[32]:=* **??Take**

```
Take[list, n] gives the first n elements of list. Take[list, -n] gives the last n elements of
   list. Take[list, {m, n}] gives elements m through n of list. Take[list, {m, n,
   s}] gives elements m through n in steps of s. Take[list, seq1, seq2, ... ] gives a
   nested list in which elements specified by seqi are taken at level i in list. Mehr...

Attributes[Take] = {NHoldRest, Protected}
```

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

```
First[expr] gives the first element in expr. Mehr...

Attributes[First] = {Protected}
```

*In[34]:=* **??Last**

```
Last[expr] gives the last element in expr. Mehr...

Attributes[Last] = {Protected}
```

*In[35]:=* **??Part**

```
expr[[i]] or Part[expr, i] gives the ith part of expr. expr[[-i]]
   counts from the end. expr[[0]] gives the head of expr. expr[[i, j, ... ]]
   or Part[expr, i, j, ... ] is equivalent to expr[[i]] [[j]] ... . expr[[
   {i1, i2, ... } ]] gives a list of the parts i1, i2, ...  of expr. Mehr...

Attributes[Part] = {NHoldRest, Protected, ReadProtected}
```

*In[36]:=* **??Rest**

```
Rest[expr] gives expr with the first element removed. Mehr...

Attributes[Rest] = {Protected}
```

*In[37]:=* **??Select**

```
Select[list, crit] picks out all elements ei of list for which crit[ei] is True. Select[
   list, crit, n] picks out the first n elements for which crit[ei] is True. Mehr...

Attributes[Select] = {Protected}
```

## 6.2.2. Anwendungen
## ▮ Applications

Hier einige Beispiele:
▮ Voici quelques exemples:

*In[38]:=* **listA = Table[Random[Integer, {0, 12}],{16}]**

*Out[38]=* {3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

*In[39]:=* **Sort[listA]**

*Out[39]=* {0, 1, 1, 1, 2, 2, 3, 3, 3, 5, 6, 6, 8, 10, 11, 12}

*In[40]:=* **Union[listA]**

*Out[40]=* {0, 1, 2, 3, 5, 6, 8, 10, 11, 12}

"Sort" sortiert die Liste, "Union" sortiert sie auch, eliminiert aber Duplikate von Elementen.
■ "Sort" trie la liste, "Union" la trie aussi, mais élimine les doubles des éléments.

```
In[41]:= Reverse[listA]

Out[41]= {1, 10, 3, 11, 2, 3, 2, 8, 6, 6, 1, 1, 12, 5, 0, 3}

In[42]:= RotateLeft[%,3]

Out[42]= {11, 2, 3, 2, 8, 6, 6, 1, 1, 12, 5, 0, 3, 1, 10, 3}

In[43]:= RotateRight[%,4]

Out[43]= {3, 1, 10, 3, 11, 2, 3, 2, 8, 6, 6, 1, 1, 12, 5, 0}

In[44]:= Permutations[{a, b, c}]

Out[44]= {{17, b, c}, {17, c, b}, {b, 17, c}, {b, c, 17}, {c, 17, b}, {c, b, 17}}

In[45]:= Permutations[{a, b, b}]

Out[45]= {{17, b, b}, {b, 17, b}, {b, b, 17}}
```

# 6.3. Erweiterung und Verkürzung von Listen
## ■ Allonger et raccourcir des listes

### 6.3.1. Befehle
#### ■ Ordres

Probiere aus: ■ Essaie:

```
In[46]:= ?Append

        Append[expr, elem] gives expr with elem appended. Mehr...

In[47]:= ?AppendTo

        AppendTo[s, elem] appends elem to the value of s, and resets s to the result. Mehr...

In[48]:= ?Drop

        Drop[list, n] gives list with its first n elements dropped. Drop[list,
           -n] gives list with its last n elements dropped. Drop[list, {n}] gives list
           with its nth element dropped. Drop[list, {m, n}] gives list with elements m
           through n dropped. Drop[list, {m, n, s}] gives list with elements m through
           n in steps of s dropped. Drop[list, seq1, seq2, ... ] gives a nested list in
           which elements specified by seqi have been dropped at level i in list. Mehr...

In[49]:= ?Insert

        Insert[list, elem, n] inserts elem at position n in list. If n is negative,
           the position is counted from the end. Insert[expr, elem, {i, j, ... }]
           inserts elem at position {i, j, ... } in expr. Insert[expr, elem, {{i1,
           j1, ... }, {i2, j2, ... }, ... }] inserts elem at several positions. Mehr...
```

*In[50]:=* **?Prepend**

```
Prepend[expr, elem] gives expr with elem prepended. Mehr...
```

*In[51]:=* **?PrependTo**

```
PrependTo[s, elem] prepends elem to the value of s, and resets s to the result. Mehr...
```

*In[52]:=* **?Rest**

```
Rest[expr] gives expr with the first element removed. Mehr...
```

*In[53]:=* **?Take**

```
Take[list, n] gives the first n elements of list. Take[list, -n] gives the last n elements of
   list. Take[list, {m, n}] gives elements m through n of list. Take[list, {m, n,
   s}] gives elements m through n in steps of s. Take[list, seq1, seq2, ... ] gives a
   nested list in which elements specified by seqi are taken at level i in list. Mehr...
```

*In[54]:=* **?First**

```
First[expr] gives the first element in expr. Mehr...
```

*In[55]:=* **?Last**

```
Last[expr] gives the last element in expr. Mehr...
```

*In[56]:=* **?Part**

```
expr[[i]] or Part[expr, i] gives the ith part of expr. expr[[-i]]
   counts from the end. expr[[0]] gives the head of expr. expr[[i, j, ... ]]
   or Part[expr, i, j, ... ] is equivalent to expr[[i]] [[j]] ... . expr[[
   {i1, i2, ... } ]] gives a list of the parts i1, i2, ...  of expr. Mehr...
```

*In[57]:=* **?Select**

```
Select[list, crit] picks out all elements ei of list for which crit[ei] is True. Select[
   list, crit, n] picks out the first n elements for which crit[ei] is True. Mehr...
```

## 6.3.2. Beispiele
## ▮ Exemples

Probiere aus: ▮ Essaie:

*In[58]:=*
```
Print[listA];
Rest[listA]
```

```
{3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}
```

*Out[59]=* {0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

Was ist passiert?
▮ Que s'est-il passé?

*In[60]:=*
```
Drop[listA, 10]
```

*Out[60]=* {3, 2, 11, 3, 10, 1}

*In[61]:=* **Drop[listA, -5]**

*Out[61]=* {3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3}

*In[62]:=* **Drop[listA, {2}]**

*Out[62]=* {3, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

*In[63]:=* **Drop[listA, {1}]**

*Out[63]=* {0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

*In[64]:=* **Drop[listA, {3}]**

*Out[64]=* {3, 0, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

*In[65]:=* **Drop[listA, {1,8}]**

*Out[65]=* {8, 2, 3, 2, 11, 3, 10, 1}

*In[66]:=* **Drop[listA, {4,12}]**

*Out[66]=* {3, 0, 5, 11, 3, 10, 1}

*In[67]:=* **Take[listA,4]**

*Out[67]=* {3, 0, 5, 12}

*In[68]:=* **Take[listA,-6]**

*Out[68]=* {3, 2, 11, 3, 10, 1}

*In[69]:=* **Take[listA,{4,6}]**

*Out[69]=* {12, 1, 1}

*In[70]:=* **Append[listA, 99]**

*Out[70]=* {3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1, 99}

*In[71]:=* **Prepend[listA, 100]**

*Out[71]=* {100, 3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

*In[72]:=* **Insert[listA, 999, 4]**

*Out[72]=* {3, 0, 5, 999, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

*In[73]:=* **listA**

*Out[73]=* {3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

Bei all diesen Operationen ist "listA" unverändert geblieben. Nicht aber bei den folgenden Operationen:

■ Pendant toutes ces opérations "listA" n'a pas changé. Cela n'est pas le cas pour les opérations suivantes:

*In[74]:=*
```
Print[listA];
AppendTo[listA, 2000]
```

{3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1}

*Out[75]=* {3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1, 2000}

*In[76]:=* **PrependTo[listA, 555]**

*Out[76]=* {555, 3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1, 2000}

---

# 6.4. Zählung der Elemente einer Liste
# ▮ Compter les éléments d'une liste

## 6.4.1. Befehle:
## ▮ Ordres:

Probiere aus: ▮ Essaie:

*In[77]:=* **?Length**

Length[expr] gives the number of elements in expr. Mehr...

*In[78]:=* **?Dimensions**

Dimensions[expr] gives a list of the dimensions of expr. Dimensions[
expr, n] gives a list of the dimensions of expr down to level n. Mehr...

## 6.4.2. Anwendungen
## ▮ Applications

*In[79]:=* **Length[listA]**

*Out[79]=* 18

*In[80]:=* **Length[{a, 4, 2^8, t^2 + 2t - 6, matrix}]**

*Out[80]=* 5

*In[81]:=* **Length[{{a, 4, 2^8, t^2 + 2t - 6, matrix},
listA}]**

*Out[81]=* 2

*In[82]:=* **Dimensions[{{a, 4, 2^8, t^2 + 2t - 6, matrix},
listA}]**

*Out[82]=* {2}

*In[83]:=* **Dimensions[{{a, 4, 2^8, t^2 + 2t - 6, matrix},
listA}, 0]**

*Out[83]=* {}

*In[84]:=* **Dimensions[{{a, 4, 2^8, t^2 + 2t - 6, matrix},
listA}, 1]**

*Out[84]=* {2}

```
In[85]:= Dimensions[{{a, 4, 2^8, t^2 + 2t - 6, matrix},
                     listA}, 2]

Out[85]= {2}

In[86]:= MatrixForm[{{a, 4, 2^8, t^2 + 2t - 6, matrix},
                     listA}]

Out[86]//MatrixForm=
         ( {17, 4, 256, -6 + 2 t + t^2, matrix}                                  )
         ( {555, 3, 0, 5, 12, 1, 1, 6, 6, 8, 2, 3, 2, 11, 3, 10, 1, 2000} )

In[87]:= Dimensions[{{a, 4, 2^8, t^2 + 2t - 6},
                     {1, 2, 3, 4}}, 2]

Out[87]= {2, 4}

In[88]:= MatrixForm[{{a, 4, 2^8, t^2 + 2t - 6},
                     {1, 2, 3, 4}}]

Out[88]//MatrixForm=
         ( 17  4  256  -6 + 2 t + t^2 )
         ( 1   2  3    4              )
```

# 6.5. Zusammenfügen von Listen, Beziehungen zwischen Listen
## ∎ Assemblage de listes, relations entre listes

### 6.5.1. Befehle
#### ∎ Ordres

Probiere aus: ∎ Essaie

```
In[89]:= ?Complement

         Complement[eall, e1, e2, ... ] gives the elements in eall which are not in any of the ei.
          Mehr…

In[90]:= ?Union

         Union[list1, list2, ... ] gives a sorted list of all the distinct
            elements that appear in any of the listi. Union[list] gives a sorted version
            of a list, in which all duplicated elements have been dropped. Mehr…

In[91]:= ?Join

         Join[list1, list2, ... ] concatenates lists together. Join
            can be used on any set of expressions that have the same head. Mehr…

In[92]:= ?Intersection

         Intersection[list1, list2, ... ] gives a sorted list of the elements common to all the listi.
          Mehr…
```

### 6.5.2. Anwendungen
### ▮ Applications

```
In[93]:= Complement[{1, 2, 3, 4, 5},{1, 2, 3}]

Out[93]= {4, 5}

In[94]:= Intersection[{a, b, c, d, e, f},
                      {d, e, f, g, h, i, j, k}]

Out[94]= {d, e, f}

In[95]:= Union[{a, b, c, d, e, f},
                                {d, e, f, g, h, i, j, k}]

Out[95]= {17, b, c, d, e, f, g, h, i, j, k}

In[96]:= Union[{1, 1, 1, 1, 2, 1, 3, 3, 3, 2, 4, 2, 4, 4,
                      3, 3, 3}]

Out[96]= {1, 2, 3, 4}

In[97]:= Join[{a, b, c, d, e, f},{d, e, f, g, h, i, j, k}]

Out[97]= {17, b, c, d, e, f, d, e, f, g, h, i, j, k}

In[98]:= Join[{1, 2, 1, 1, 2, 2},{1, 2, 2, 1, 3, 2, 1,
                      3, 3}]

Out[98]= {1, 2, 1, 1, 2, 2, 1, 2, 2, 1, 3, 2, 1, 3, 3}

In[99]:= Union[{1, 2, 1, 1, 2, 2},{1, 2, 2, 1, 3, 2, 1,
                      3, 3}]

Out[99]= {1, 2, 3}
```

---

## 6.6. Veränderung der Form einer Liste
## ▮ Changer la forme d'une liste

### 6.6.1. Befehle
### ▮ Ordres

Probiere aus: ▮ Essaie:

```
In[100]:=
       ?Flatten

        Flatten[list] flattens out nested lists. Flatten[list, n] flattens
           to level n. Flatten[list, n, h] flattens subexpressions with head h. Mehr...
```

*In[101]:=*

```
?Partition
```

```
Partition[list, n] partitions list into non-overlapping sublists of length n. Partition[list,
   n, d] generates sublists with offset d. Partition[list, {n1, n2, ... }] partitions
   a nested list into blocks of size n1 × n2 × … . Partition[list, {n1, n2, ... },
   {d1, d2, ... }] uses offset di at level i in list. Partition[list, n, d, {kL, kR}]
   specifies that the first element of list should appear at position kL in the first
   sublist, and the last element of list should appear at or after position kR in the
   last sublist. If additional elements are needed, Partition fills them in by treating
   list as cyclic. Partition[list, n, d, {kL, kR}, x] pads if necessary by repeating
   the element x. Partition[list, n, d, {kL, kR}, {x1, x2, ... }] pads if necessary by
   cyclically repeating the elements xi. Partition[list, n, d, {kL, kR}, {}] uses no
   padding, and so can yield sublists of different lengths. Partition[list, nlist, dlist,
   {klistL, klistR}, padlist] specifies alignments and padding in a nested list. Mehr...
```

*In[102]:=*

```
?Transpose
```

```
Transpose[list] transposes the first two levels in list.
   Transpose[list, {n1, n2, ... }] transposes list so that the levels 1,
   2, ... in list correspond to levels n1, n2, ... in the result. Mehr...
```

## 6.6.2. Anwendungen
## ▮ Applications

*In[103]:=*

```
vector1 = Range[10]
```

*Out[103]=*

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

*In[104]:=*

```
matrix1 = Partition[vector1, 5]
```

*Out[104]=*

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

*In[105]:=*

```
MatrixForm[matrix1]
```

*Out[105]//MatrixForm=*

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{pmatrix}$$

*In[106]:=*

```
matrix2 = Partition[vector1, 4]
```

*Out[106]=*

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

*In[107]:=*

```
matrix3 = Partition[vector1, 3]
```

*Out[107]=*

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

*In[108]:=*
```
matrix4 = Partition[vector1, 2.5]
```

```
Partition::ilsmp : Single or list of positive machine-size integers
   expected at position 2 of Partition[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, 2.5]. Mehr…
```

*Out[108]=*
```
Partition[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, 2.5]
```

Was ist passiert?

■ Que s'est-il passé?

*In[109]:=*
```
Partition[{1, 35, 432, 2, 5467, 4, 3, 987, 87},
                                              3]
```

*Out[109]=*
```
{{1, 35, 432}, {2, 5467, 4}, {3, 987, 87}}
```

*In[110]:=*
```
Partition[{1, 2, 3, 4, 5, 6, 7}, 3, 1]
```

*Out[110]=*
```
{{1, 2, 3}, {2, 3, 4}, {3, 4, 5}, {4, 5, 6}, {5, 6, 7}}
```

*In[111]:=*
```
Partition[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9},
            {a, b, c}}, 2, 1]
```

*Out[111]=*
```
{{{1, 2, 3}, {4, 5, 6}}, {{4, 5, 6}, {7, 8, 9}}, {{7, 8, 9}, {17, b, c}}}
```

*In[112]:=*
```
Print[matrix1];
Flatten[matrix1]
```

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

*Out[113]=*
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

*In[114]:=*
```
m = {{1, 2, 3, a},{4, 5, 6, b}}
```

*Out[114]=*
```
{{1, 2, 3, 17}, {4, 5, 6, b}}
```

*In[115]:=*
```
Transpose[m]
```

*Out[115]=*
```
{{1, 4}, {2, 5}, {3, 6}, {17, b}}
```

In[116]:=
```
Print[MatrixForm[m]];
MatrixForm[Transpose[m]]
```

$$\begin{pmatrix} 1 & 2 & 3 & 17 \\ 4 & 5 & 6 & b \end{pmatrix}$$

Out[117]//MatrixForm=

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \\ 17 & b \end{pmatrix}$$

# 6.7. Elemente herauspicken
## Choisir quelques éléments

### 6.7.1. Befehle
#### Ordres

Oben sind schon "First", "Last" und "Part erwähnt worden.

- Plus haut on a déjà nommé "First", "Last", et "Post".

### 6.7.2. Anwendungen
#### Applications

In[118]:=
```
liste1 = {a, b, c, d, e, f, g}
```

Out[118]=
```
{17, b, c, d, e, f, g}
```

In[119]:=
```
First[liste1]
```

Out[119]=
```
17
```

In[120]:=
```
Last[liste1]
```

Out[120]=
```
g
```

In[121]:=
```
liste1[[4]]
```

Out[121]=
```
d
```

In[122]:=
```
Print[{liste1[[2]],liste1[[3]],liste1[[4]],
        liste1[[5]]}]
```

```
{b, c, d, e}
```

*In[123]:=*
```
liste1[[ Range[2, 5] ]]
```
*Out[123]=*
```
{b, c, d, e}
```

*In[124]:=*
```
array1 = Array[a, {2, 2}]
```
*Out[124]=*
```
{{17[1, 1], 17[1, 2]}, {17[2, 1], 17[2, 2]}}
```

*In[125]:=*
```
array1[[2]]
```
*Out[125]=*
```
{17[2, 1], 17[2, 2]}
```

*In[126]:=*
```
array1[[1, 2]]
```
*Out[126]=*
```
17[1, 2]
```

---

# 6.8. Auswahl von Daten
## ▮ Choisir des données

### 6.8.1. Befehle
#### ▮ Ordres

Probiere aus: ▮ Essaie:

*In[127]:=*
```
?Select
```
```
Select[list, crit] picks out all elements ei of list for which crit[ei] is True. Select[
   list, crit, n] picks out the first n elements for which crit[ei] is True. Mehr...
```

In[128]:=
```
?*Q
```

**System`**

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

In[129]:=
```
?DigitQ
```

DigitQ[string] yields True if all the characters in the string
   are digits in the range 0 through 9, and yields False otherwise. Mehr...

In[130]:=
```
?IntegerQ
```

IntegerQ[expr] gives True if expr is an integer, and False otherwise. Mehr...

In[131]:=
```
?LetterQ
```

LetterQ[string] yields True if all the characters
   in the string are letters, and yields False otherwise. Mehr...

In[132]:=
```
?LowerCaseQ
```

LowerCaseQ[string] yields True if all the characters in
   the string are lower-case letters, and yields False otherwise. Mehr...

In[133]:=
```
?MachineNumberQ
```

MachineNumberQ[expr] returns True if expr is a machine-
   precision real or complex number, and returns False otherwise. Mehr...

*In[134]:=*
**?MatrixQ**

MatrixQ[expr] gives True if expr is a list of lists or a two-dimensional SparseArray object that can represent a matrix, and gives False otherwise. MatrixQ[expr, test] gives True only if test yields True when applied to each of the matrix elements in expr. Mehr...

*In[135]:=*
**?NameQ**

NameQ["string"] yields True if there are any symbols whose names match the string pattern given, and yields False otherwise. Mehr...

*In[136]:=*
**?NumberQ**

NumberQ[expr] gives True if expr is a number, and False otherwise. Mehr...

*In[137]:=*
**?OddQ**

OddQ[expr] gives True if expr is an odd integer, and False otherwise. Mehr...

*In[138]:=*
**?EvenQ**

EvenQ[expr] gives True if expr is an even integer, and False otherwise. Mehr...

*In[139]:=*
**?PrimeQ**

PrimeQ[expr] yields True if expr is a prime number, and yields False otherwise. Mehr...

*In[140]:=*
**?UpperCaseQ**

UpperCaseQ[string] yields True if all the characters in the string are upper-case letters, and yields False otherwise. Mehr...

*In[141]:=*
**?ValueQ**

ValueQ[expr] gives True if a value has been defined for expr, and gives False otherwise. Mehr...

*In[142]:=*
**?VectorQ**

VectorQ[expr] gives True if expr is a list or a one-dimensional SparseArray object, none of whose elements are themselves lists, and gives False otherwise. VectorQ[expr, test] gives True only if test yields True when applied to each of the elements in expr. Mehr...

*In[143]:=*
**?Positiv***

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

*In[144]:=*
**?Negat***

Negative[x] gives True if x is a negative number. Mehr...

### 6.8.2. Anwendungen
### ▮ Applications

Probiere aus: ▮ Essaie:

*In[145]:=*
```
MemberQ[liste1, 3]
```

*Out[145]=*
```
False
```

*In[146]:=*
```
MemberQ[liste1, a]
```

*Out[146]=*
```
True
```

*In[147]:=*
```
menge = {-4, 3, 5.678, emil, Emma, 4, 0, -2, -4,
            Pi, 2^(1/2)}
```

*Out[147]=*
$$\left\{-4,\ 3,\ 5.678,\ \text{emil},\ \text{Emma},\ 4,\ 0,\ -2,\ -4,\ \pi,\ \sqrt{2}\right\}$$

*In[148]:=*
```
Select[menge, Positive]
```

*Out[148]=*
$$\left\{3,\ 5.678,\ 4,\ \pi,\ \sqrt{2}\right\}$$

*In[149]:=*
```
Select[{a, b, c, d, x, y, z}, Positive]
```

*Out[149]=*
```
{17}
```

Bei "Select" steht also hinten das Selektionskriterium. Solche Kriterien kann man selbst definieren. Beispiel:
▮ Quant à "Select", le critère de sélection se trouve à la fin. On peut définir soi-même de tels critères. Exemple:

*In[150]:=*
```
?Precision
```

```
Precision[x] gives the effective number of digits of precision in the number x. Mehr…
```

*In[151]:=*
```
?===
```

```
lhs === rhs yields True if the expression
   lhs is identical to rhs, and yields False otherwise. Mehr…
```

*In[152]:=*
```
exakt[x_] := Precision[x] === Infinity
```

*In[153]:=*
```
Select[{-6, 0, 3, 3.1414333, 1.111111, Pi, E,
        N[Sqrt[2], 10], 5.66}, exakt]
```

*Out[153]=*
$$\{-6,\ 0,\ 3,\ \pi,\ e\}$$

# 6.9. Rechnen mit Listen
## ▮ Calculer avec des listes

### 6.9.1. Befehle
#### ▮ Ordres

*In[154]:=*
```
?Sum
```

```
Sum[f, {i, imax}] evaluates the sum of the expressions f as
   evaluated for each i from 1 to imax. Sum[f, {i, imin, imax}] starts with
   i = imin. Sum[f, {i, imin, imax, di}] uses steps di. Sum[f, {i, imin,
   imax}, {j, jmin, jmax}, ... ] evaluates a sum over multiple indices. Mehr...
```

*In[155]:=*
```
?Plus
```

```
x + y + z represents a sum of terms. Mehr...
```

*In[156]:=*
```
?Apply
```

```
Apply[f, expr] or f @@ expr replaces the head of expr by f. Apply[f, expr,
   levelspec] replaces heads in parts of expr specified by levelspec. Mehr...
```

### 6.9.2. Wie man's macht:
#### ▮ C'est ainsi que cela se fait:

*In[157]:=*
```
Sum[n^2, {n, 7}]
```

*Out[157]=*
```
140
```

*In[158]:=*
```
meineDaten = {3.4, 6.1, 5.3}
```

*Out[158]=*
```
{3.4, 6.1, 5.3}
```

*In[159]:=*
```
Sum[meineDaten[[n]], {n, Length[meineDaten]}]
```

*Out[159]=*
```
14.8
```

*In[160]:=*
```
Plus[1, 1, 1, 1, 1]
```

*Out[160]=*
```
5
```

*In[161]:=*
```
Plus[1, 2, 3, 4, 5]
```
*Out[161]=*
```
15
```

*In[162]:=*
```
Plus[a, b, c, d]
```
*Out[162]=*
```
17 + b + c + d
```

*In[163]:=*
```
Plus[{a, b, c, d}]
```
*Out[163]=*
```
{17, b, c, d}
```

*In[164]:=*
```
??Plus
```
```
x + y + z represents a sum of terms. Mehr...

Attributes[Plus] = {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}

Default[Plus] := 0
```

"Plus" hat das Attribut "Listable".

■ "Plus" a l'attribut "Listable".

*In[165]:=*
```
??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...

Attributes[Listable] = {Protected}
```

"Plus" auf eine Liste angewandt addiert nur ein Element, nämlich die Liste selbst. Wie nun "Plus" aber auf die Elemente anwenden? - So:

■ "Plus" appliqué à une liste additionne seulement un élément, c'est-à-dire la liste même. Comment appliquer "Plus" aux éléments? - Ainsi:

*In[166]:=*
```
Apply[Plus, {a, b, c, d}]
```
*Out[166]=*
```
17 + b + c + d
```

Was geht nun schneller, "Sum" oder "Plus" ?

■ Qu'est-ce qui est plus rapide, "Sum" ou "Plus"?

*In[167]:=*
```
Timing[Apply[Plus, Range[500]]]
```
*Out[167]=*
```
{0. Second, 125250}
```

In[168]:=
```
Timing[Sum[n, {n, 500}]]
```

Out[168]=
```
{0. Second, 125250}
```

## 6.10. Auf Listen anwendbare arithmetische Funktionen
## ▮ Fonctions arithmétiques applicables à des listes

### Beispiele
### ▮ Exemples

Wir wenden "Log" auf eine Liste an:
▮ Nous appliquons "Log" à une liste:

In[169]:=
```
??Log
```

```
Log[z] gives the natural logarithm of z (
   logarithm to base e). Log[b, z] gives the logarithm to base b. Mehr...

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

In[170]:=
```
Log[{1, 2, 3, 4, 5, 6, 7}]
```

Out[170]=
```
{0, Log[2], Log[3], Log[4], Log[5], Log[6], Log[7]}
```

In[171]:=
```
N[%]
```

Out[171]=
```
{0., 0.693147, 1.09861, 1.38629, 1.60944, 1.79176, 1.94591}
```

Oder mit "Sinus":
▮ Ou avec "Sinus":

In[172]:=
```
Sin[{1, 2, 3, 4, 5, 6, 7}]
```

Out[172]=
```
{Sin[1], Sin[2], Sin[3], Sin[4], Sin[5], Sin[6], Sin[7]}
```

In[173]:=
```
Sin[{1, 2, 3, 4, 5, 6, 7}] // N
```

Out[173]=
```
{0.841471, 0.909297, 0.14112, -0.756802, -0.958924, -0.279415, 0.656987}
```

Oder "Plus":
▮ Ou "Plus":

*In[174]:=*
```
?Plus
```

x + y + z represents a sum of terms. Mehr...

*In[175]:=*
```
?*Form*
```

**System`**

| | |
|---|---|
| AccountingForm | MathMLForm |
| BaseForm | MatrixForm |
| BinaryFormat | NumberForm |
| BlankForm | NumberFormat |
| BoxFormFormatTypes | OutputForm |
| CForm | OutputFormData |
| ColonForm | OutputToOutputForm |
| ColumnForm | OutputToStandardForm |
| CommonDefaultFormatTypes | PaddedForm |
| DefaultFormatType | ParentForm |
| DefaultFormatTypeForStyle | PasteBoxFormInlineCells |
| DefaultInlineFormatType | PointForm |
| DefaultInputFormatType | PolynomialForm |
| DefaultOutputFormatType | PrecedenceForm |
| DefaultTextFormatType | PrintForm |
| DefaultTextInlineFormatType | PromptForm |
| DisplayForm | RealBlockForm |
| DOSTextFormat | ReturnInputFormPacket |
| EdgeForm | RuleForm |
| EngineeringForm | ScientificForm |
| ExcludedForms | SequenceForm |
| FaceForm | SetBoxFormNamesPacket |
| FileFormat | ShowShortBoxForm |
| FontForm | SpaceForm |
| Format | StandardForm |
| FormatRules | StringForm |
| FormatType | StyleForm |
| FormatTypeAutoConvert | SyntaxForm |
| FormatValues | TableForm |
| FormBox | TeXForm |
| FormBoxOptions | TextForm |
| FortranForm | TraditionalForm |
| FullForm | TreeForm |
| HoldForm | ValueForm |
| HorizontalForm | VerticalForm |
| InputAutoFormat | $BoxForms |
| InputForm | $ExportFormats |
| InputToBoxFormPacket | $FormatType |
| InputToInputForm | $ImportFormats |
| InputToStandardForm | $OutputForms |
| LineForm | $PrintForms |
| LongForm | $SuppressInputFormHeads |

```
In[176]:=
FullForm[a + b]

Out[176]//FullForm=
Plus[17, b]

In[177]:=
Plus[{1, 2, 3, 4}, {a, b, c, d}]

Out[177]=
{18, 2 + b, 3 + c, 4 + d}

In[178]:=
{1, 2, 3, 4} + {a, b, c, d}

Out[178]=
{18, 2 + b, 3 + c, 4 + d}
```

Attribute einiger Funktionen:

■ Attributs de quelques fonctions:

```
In[179]:=
Attributes[{Exp, Log, Sin, Cos, Plus, Times}]

Out[179]=
{{Listable, NumericFunction, Protected, ReadProtected},
 {Listable, NumericFunction, Protected},
 {Listable, NumericFunction, Protected}, {Listable, NumericFunction, Protected},
 {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected},
 {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}}

In[180]:=
Attributes[Attributes]

Out[180]=
{HoldAll, Listable, Protected}
```

"Attributes" hat also auch das Attribut "Listable".

■ "Attributes" a donc aussi l'attribut "Listable".

---

# 6.11. Listable und Map
## ■ Listable et Map

### 6.11.1. Befehle
#### ■ Ordres

Probiere: ■ Essaie:

*In[181]:=*
```
??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[182]:=*
```
?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[183]:=*
```
?Apply
```

```
Apply[f, expr] or f @@ expr replaces the head of expr by f. Apply[f, expr,
   levelspec] replaces heads in parts of expr specified by levelspec. Mehr...
```

## 6.11.2. Anwendungen
### ▮ Applications

Es gibt zwei Wege, eine Funktion auf eine Liste anzuwenden: mit "Listable" als Attribut und mit der Funktion "Map". Probiere:

▮ Il y a deux façons d'appliquer une fonction à une liste: avec l'attribut "List" et avec la fonction "Map". Essaie:

*In[184]:=*
```
f[{1, 2, 3, 4, 5, 6}]
```

*Out[184]=*
```
f[{1, 2, 3, 4, 5, 6}]
```

*In[185]:=*
```
?f
```

```
Global`f
```

f ist nicht "Listable".

▮ F n'est pas "Listable"

*In[186]:=*
```
Map[f, {1, 2, 3, 4, 5, 6}]
```

*Out[186]=*
```
{f[1], f[2], f[3], f[4], f[5], f[6]}
```

So geht's! Oder so:

▮ Voilà! Ou ainsi:

*In[187]:=*
```
Map[f, a x^2 + b x + c]
```

*Out[187]=*
```
f[c] + f[b x] + f[17 x^2]
```

Map wirkt auf die Teile eines Ausdrucks! Beachte:

■ Map a un effet sur les parties d'une expression! Remarque:

*In[188]:=*
```
Map[f, {a, b, c, d, e}]
```
*Out[188]=*
```
{f[17], f[b], f[c], f[d], f[e]}
```

*In[189]:=*
```
Apply[f, {a, b, c, d, e}]
```
*Out[189]=*
```
f[17, b, c, d, e]
```

Mit Apply wird f angewandt auf die Elemente der Menge als Ganzes, während mit Map diejenige Menge gebildet wird, die als neue Elemente die Funktionswerte der einzelnen alten Elemente enthält.

■ Avec Apply on applique f aux éléments de l'ensemble comme un tout, tandis qu'avec Map on forme l'ensemble qui contient en tant qu'éléments nouveaux les valeurs des fonctions des vieux éléments.

Statt Apply und Map kann man auch /@ und @@ benutzen. Beispiele:

■ On peut employer aussi /@ et @@ au lieu de Apply et Map. Exemple:

*In[190]:=*
```
f/@{a, b, c, d, e}
```
*Out[190]=*
```
{f[17], f[b], f[c], f[d], f[e]}
```

*In[191]:=*
```
f@@{a, b, c, d, e}
```
*Out[191]=*
```
f[17, b, c, d, e]
```

*In[192]:=*
```
Sin/@{0, Pi, 2 Pi, 2, 3, 4, 5, 6}
```
*Out[192]=*
```
{0, 0, 0, Sin[2], Sin[3], Sin[4], Sin[5], Sin[6]}
```

*In[193]:=*
```
f1[x_, y_,z_]:= Sin[x z^2] Cos[y]
```

*In[194]:=*
```
f1@@{1, 2, 3}
```
*Out[194]=*
```
Cos[2] Sin[9]
```

# 6.12. Formatierung von Listen
## ▮ Formater des listes

### 6.12.1. Befehle
#### ▮ Ordres

Probiere: ▮ Essaie:

*In[195]:=*
```
?ColumnForm
```

```
ColumnForm[{e1, e2, ... }] prints as a column with e1 above e2, etc. ColumnForm[list,
   horiz] specifies the horizontal alignment of each element. ColumnForm[list,
   horiz, vert] also specifies the vertical alignment of the whole column. Mehr...
```

*In[196]:=*
```
?MatrixForm
```

```
MatrixForm[list] prints with the elements of list arranged in a regular array. Mehr...
```

*In[197]:=*
```
?TableForm
```

```
TableForm[list] prints with the elements of list arranged in an array of rectangular cells.
 Mehr...
```

### 6.12.2. Beispiele
#### ▮ Exemples

*In[198]:=*
```
?Binomial
```

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

*In[199]:=*
```
pascal[n_]:= Table[Binomial[n, i],{i, 0, n}]
```

*In[200]:=*
```
pascal[5]
```

*Out[200]=*
```
{1, 5, 10, 10, 5, 1}
```

*In[201]:=*
```
Table[pascal[n],{n, 0, 5}]
```

*Out[201]=*
```
{{1}, {1, 1}, {1, 2, 1}, {1, 3, 3, 1}, {1, 4, 6, 4, 1}, {1, 5, 10, 10, 5, 1}}
```

*In[202]:=*

```
ColumnForm[Table[pascal[n],{n, 0, 5}]]
```

*Out[202]=*

```
{1}
{1, 1}
{1, 2, 1}
{1, 3, 3, 1}
{1, 4, 6, 4, 1}
{1, 5, 10, 10, 5, 1}
```

*In[203]:=*

```
ColumnForm[Table[pascal[n],{n, 0, 5}], Center]
```

*Out[203]=*

```
        {1}
      {1, 1}
    {1, 2, 1}
  {1, 3, 3, 1}
{1, 4, 6, 4, 1}
{1, 5, 10, 10, 5, 1}
```

*In[204]:=*

```
ColumnForm[Table[pascal[n],{n, 0, 5}], Right]
```

*Out[204]=*

```
                {1}
              {1, 1}
            {1, 2, 1}
          {1, 3, 3, 1}
        {1, 4, 6, 4, 1}
{1, 5, 10, 10, 5, 1}
```

*In[205]:=*

```
ColumnForm[Table[pascal[n],{n, 0, 5}], Left]
```

*Out[205]=*

```
{1}
{1, 1}
{1, 2, 1}
{1, 3, 3, 1}
{1, 4, 6, 4, 1}
{1, 5, 10, 10, 5, 1}
```

*In[206]:=*

```
TableForm[Table[pascal[n],{n, 0, 5}]]
```

*Out[206]//TableForm=*

```
1
1     1
1     2     1
1     3     3     1
1     4     6     4     1
1     5     10    10    5     1
```

*In[207]:=*

```
MatrixForm[Table[pascal[n],{n, 0, 5}]]
```

*Out[207]//MatrixForm=*

$$\begin{pmatrix} \{1\} \\ \{1, 1\} \\ \{1, 2, 1\} \\ \{1, 3, 3, 1\} \\ \{1, 4, 6, 4, 1\} \\ \{1, 5, 10, 10, 5, 1\} \end{pmatrix}$$

*In[208]:=*

```
MatrixForm[Table[n - i,{i,5},{n, 5}]]
```

*Out[208]//MatrixForm=*

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ -1 & 0 & 1 & 2 & 3 \\ -2 & -1 & 0 & 1 & 2 \\ -3 & -2 & -1 & 0 & 1 \\ -4 & -3 & -2 & -1 & 0 \end{pmatrix}$$

---

# "Putzmaschine" einsetzen
# ▮ Employer la "machine de nettoyage"

*In[209]:=*

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

*In[210]:=*

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