Grundlagen der Programmierung

10. Vorlesung
19.12.2017

Funktionen

Funktionen mit Parametern

aktueller Parameter

```
n = 6;
print_n_stars( 5 );
fprintf('\d',n);

function print_n_stars( n )
    for x = 1:n
        fprintf('*');
    end
end
```

Wert von *n* wird nicht verändert!

formaler Parameter

Funktion `print_n_stars` wird mit *n*=5 ausgeführt

Call-by-Value: nur der Wert wird übergeben

Funktionen mit Rückgabe

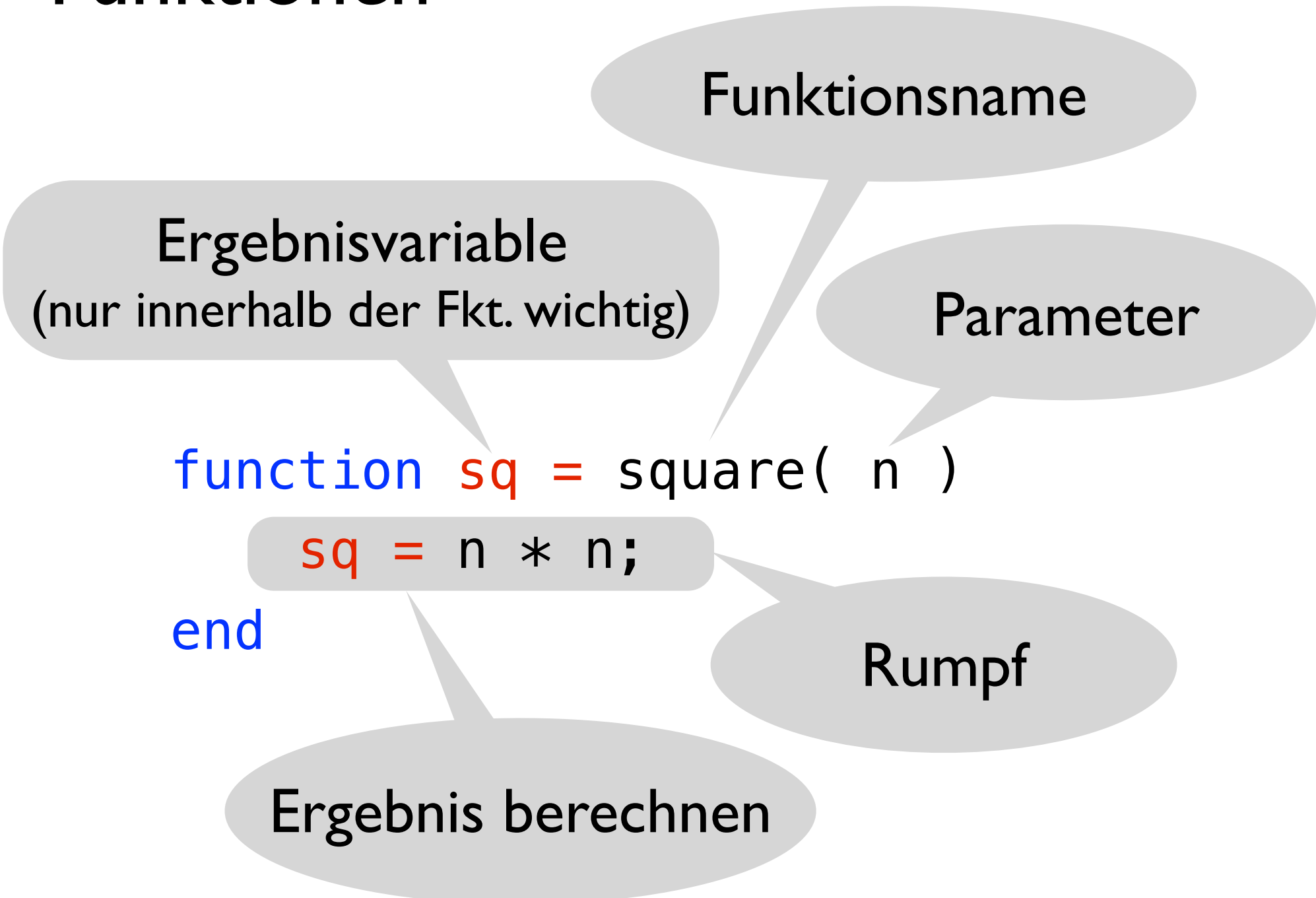
- Eingabe: x, Rückgabe: x^2

- $$z = 5 + \texttt{square}(\underbrace{2+3}_{5})$$

$$\underbrace{\underbrace{5 + \texttt{square}(5)}_{25}}_{30}$$

Wie programmiert man so etwas in Matlab ?

Funktionen


Funktionsname
Ergebnisvariable
(nur innerhalb der Fkt. wichtig)
Parameter
function sq = square(n)
sq = n * n;
end
Rumpf
Ergebnis berechnen

Funktionen: Aufruf in Ausdruck

```
n = input('Bitte Zahl eingeben: ');
m = square( n );
fprintf('Das Quadrat von %d ist %d\n', ...
          n, m );
```

Funktionsaufruf

```
n = input('Bitte Zahl eingeben: ');
fprintf('Das Quadrat von %d ist %d\n', ...
          n, square( n ) );
```

Noch einmal ganz genau

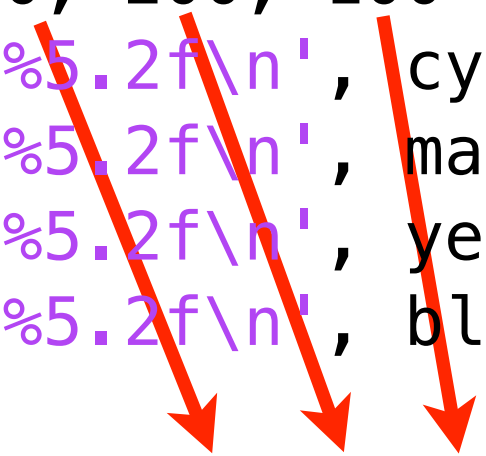
```
a = 3;
b = square( a );
```

Bei der Auswertung eines Ausdrucks wird ein Funktionsaufruf durch den Wert der Ergebnisvariablen ersetzt

```
function sq = square( n = 3 )
  sq = n * n ;
end
```

Mehrere Parameter und Rückgaben

```
[cy, ma, ye, bl] = rgb_to_cmyk( 0, 200, 100 );
fprintf('der Cyan    Anteil ist %5.2f\n', cy);
fprintf('der Magenta Anteil ist %5.2f\n', ma);
fprintf('der Yellow  Anteil ist %5.2f\n', ye);
fprintf('der Black   Anteil ist %5.2f\n', bl);

function [C, M, Y, K] = rgb_to_cmyk( R, G, B )
  Rtemp = R / 255;
  Gtemp = G / 255;
  Btemp = B / 255;
  K = 1 - max([Rtemp, Gtemp, Btemp]);
  C = (1-Rtemp-K) / (1-K);
  M = (1-Gtemp-K) / (1-K);
  Y = (1-Btemp-K) / (1-K);
end
```



Lokale / globale Funktionen

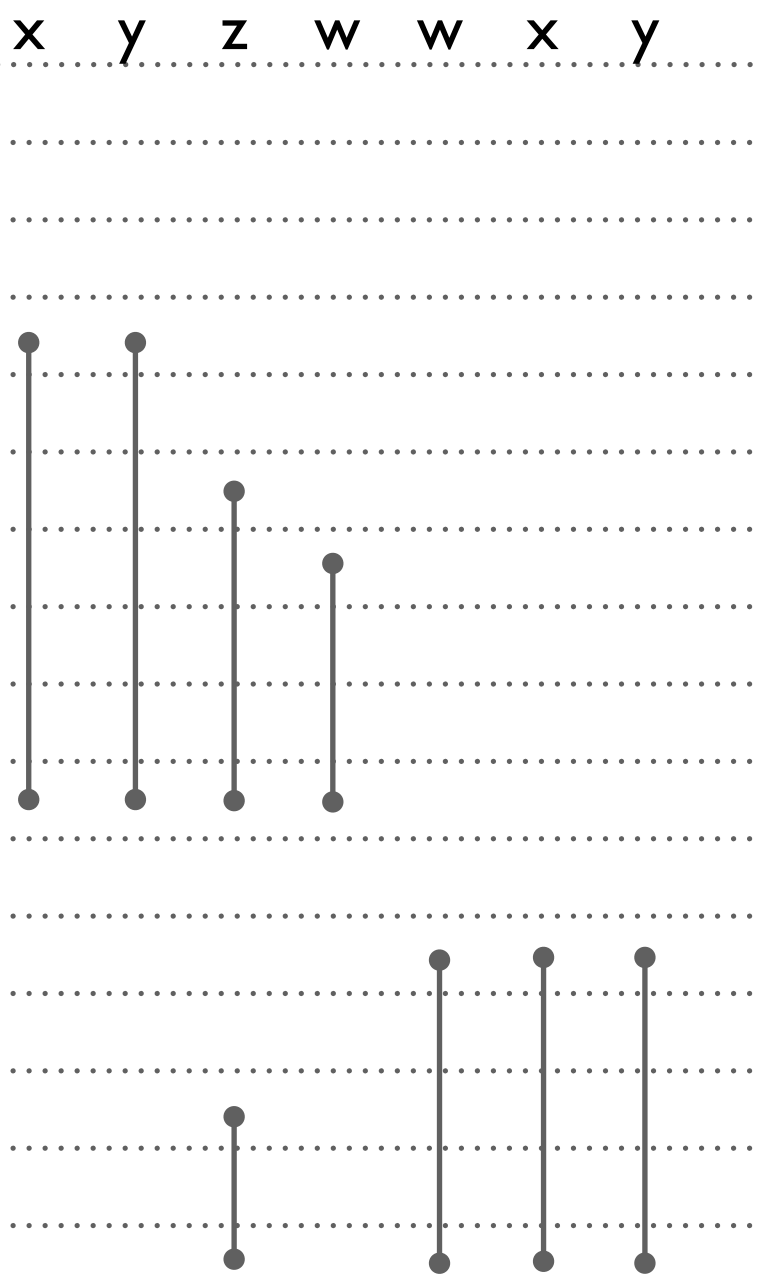
- Deklaration unter Hauptprogramm ➜ *lokale* Fkt.
  - nicht in anderen Skripten/Console verfügbar
- *Globale* Funktion:
  - `function` Deklaration in erster Zeile
  - Skriptname = Funktionsname
  - % unter `function`: Hilfetext (optional)

Sichtbarkeit von Variablen

- Jeder Funktionsaufruf hat einen eigenen Bereich zum Speichern von Variablen
- Eine Funktion kann nicht auf die Variablen einer anderen Funktion zugreifen
- Ausnahmen:
  - Deklaration mit `global`: gemeinsamer Speicherbereich für alle Funktionen
  - Deklaration mit `persistent`: gemeinsamer Speicherbereich für alle Aufrufe einer Funktion

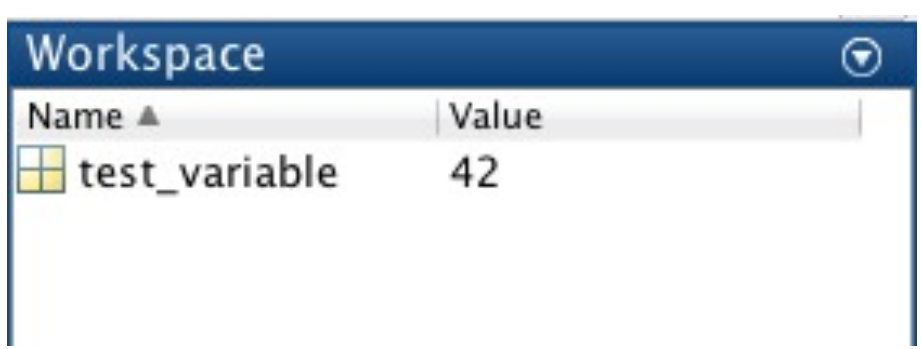
Gültigkeitsbereiche von Variablen

```
x = 42;
a(x)

function y = a( x )
    ...
    global z;
    w = z - x;
    y = b(x, w);
        ...
end

function w = b( x, y )
    ...
    global z;
    w = z + w + y;
end
```



Variablen im Hauptprogramm

- Teilen sich einen gemeinsamen Speicher (‘Workspace’), auch mit der Konsole
- Script 1 “script_set”:
```
test_variable = 42;
```
- Script 2 “script_get”:
```
fprintf('%d\n',
        test_variable);
```
- nach Script 1:




```
>> script_set
>> script_get
42
>> test_variable

test_variable =

    42

>> test_variable = 33;
>> script_get
33
>>
```

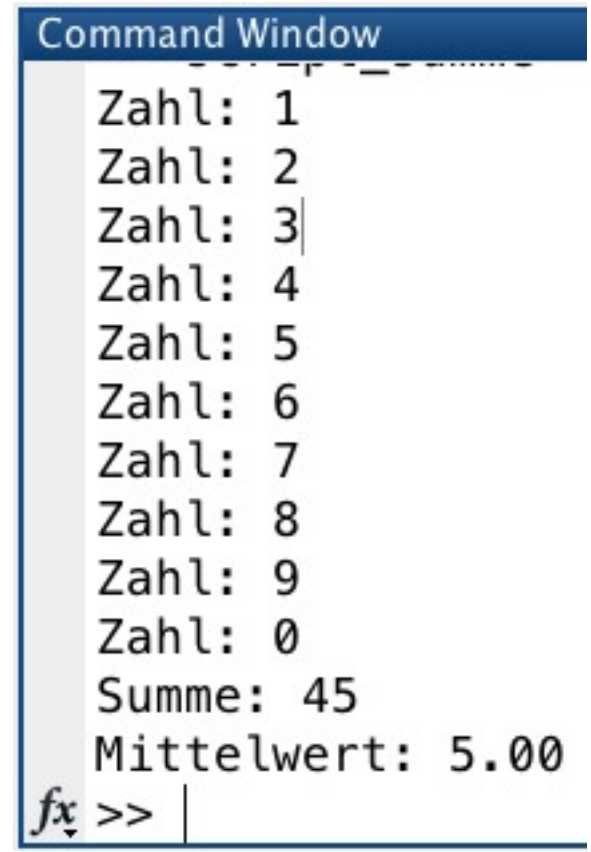
Arrays, Vektoren, Matrizen

Beispiel

- Aufgabe: Zahlen von Tastatur einlesen, bis 0 eingegeben wird, Durchschnitt berechnen
- $$\text{mean} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$
- $$\text{mean} = \frac{1}{n}\left(x_1 + x_2 + \ldots + x_n\right)$$

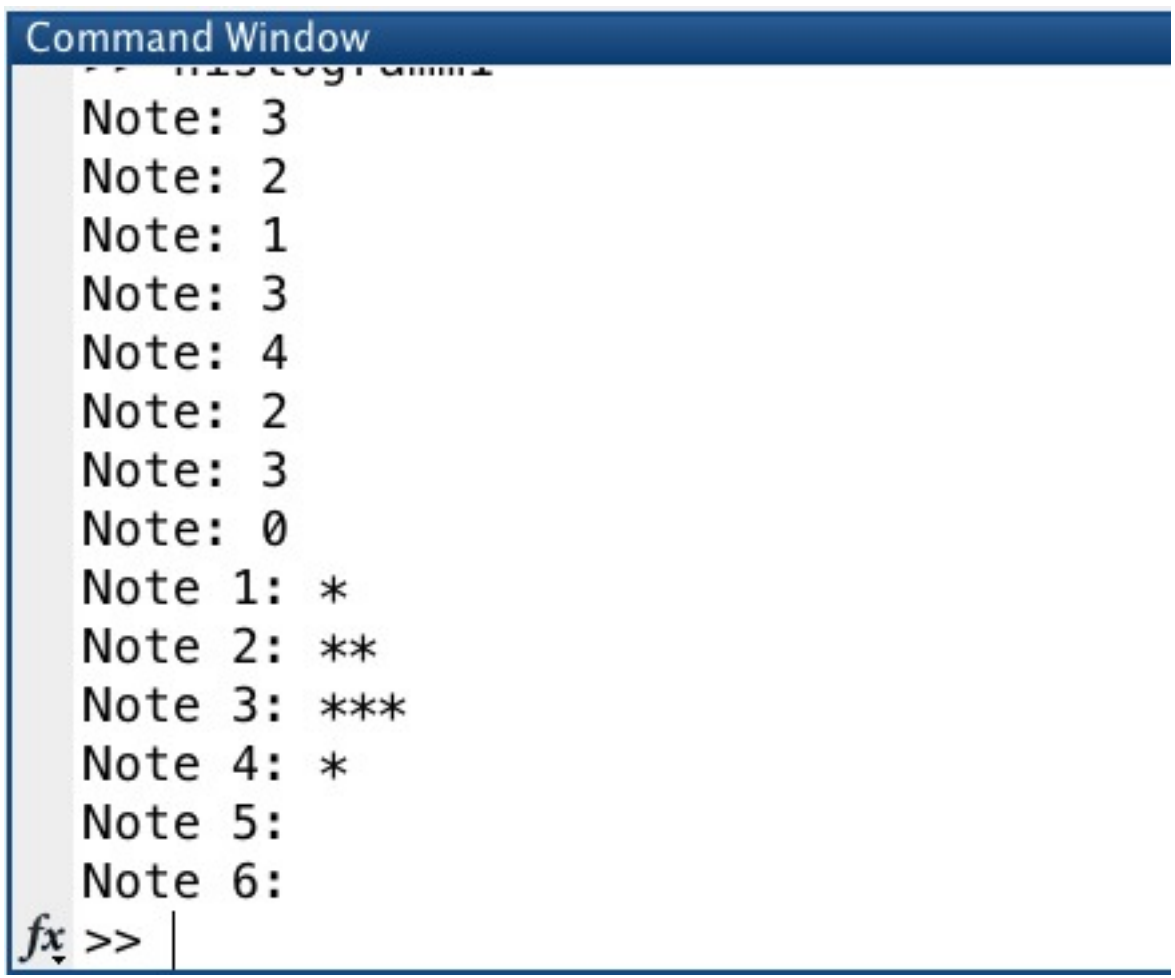
Beispiel

```
summe = 0;
n = 0;
eingabe = 1;
while eingabe ~= 0
    eingabe = input('Zahl: ');
    summe = summe + eingabe;
    n = n + 1;
end
fprintf('Summe: %d\n', summe);
fprintf('MW: %.2f\n', summe / (n-1) );
```



Beispiel: Notenhistogramm

- Aufgabe: Zahlen (Noten von 1-6) von Tastatur einlesen, bis 0 eingegeben wird, Histogramm ausgeben



n Sterne ausgeben

```
function print_n_stars( n )
% This prints n stars
    for x = 1:n
        fprintf('*');
    end
end
```

Notenhistogramm: Eingabe

```
n1 = 0; n2 = 0; n3 = 0; n4 = 0; n5 = 0; n6 = 0;
eingabe = 1;
while eingabe ~= 0
    eingabe = input('Note: ');
    switch eingabe
        case 1
            n1 = n1 + 1;
        case 2
            n2 = n2 + 1;
        case 3
            n3 = n3 + 1;
        case 4
            n4 = n4 + 1;
        case 5
            n5 = n5 + 1;
        case 6
            n6 = n6 + 1;
    end
end
```

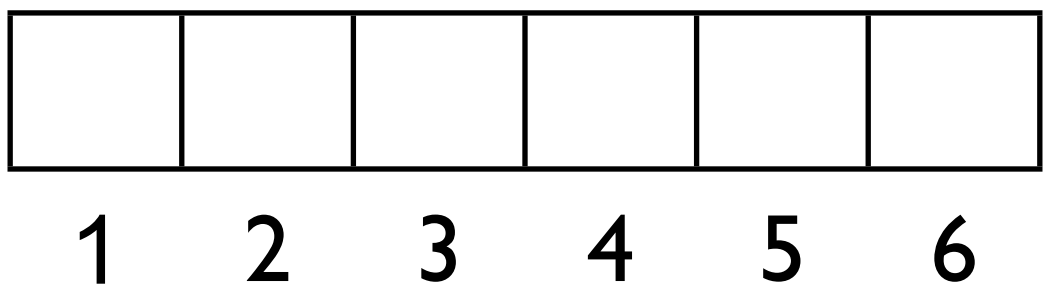
Notenhistogramm: Ausgabe

```
fprintf('\nNote 1: ');
print_n_stars( n1 );
fprintf('\nNote 2: ');
print_n_stars( n2 );
fprintf('\nNote 3: ');
print_n_stars( n3 );
fprintf('\nNote 4: ');
print_n_stars( n4 );
fprintf('\nNote 5: ');
print_n_stars( n5 );
fprintf('\nNote 6: ');
print_n_stars( n6 );
```

Geht das einfacher/ übersichtlicher?

Arrays (Felder, Vektoren)

- Container für mehrere Objekte
- Objekte in dem Container werden über *Indizes* adressiert (beginnend bei 1)



- Für Indizes dürfen Ausdrücke benutzt werden (d.h. Indizes können berechnet werden)
- In Matlab: alle Variablen sind Felder

Felder: erzeugen

- Zeilenvektoren:
  `a = [1 2 3 4]`
- Spaltenvektoren:
  `b = [5; 6; 7; 8]`
- Matrizen:
  `c = [1 2; 3 4]`
- Bereiche:
  - `d = 1:5`
  - `e = 1:2:10`

```
a =

     1     2     3     4

b =

     5
     6
     7
     8

c =

     1     2
     3     4

d =

     1     2     3     4     5

e =

     1     3     5     7     9
```

Felder: erzeugen

- Matrix mit Einsen:
  `f = ones(3,3)`

```
f =

     1     1     1
     1     1     1
     1     1     1
```

- Matrix mit Nullen:
  `g = zeros(3,3)`

```
g =

     0     0     0
     0     0     0
     0     0     0
```

- Matrix mit Zufallszahlen:
  `h = rand(3,3)`

```
h =

    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

- Äquidistanter Vektor:
  `g = linspace(1,20,7)` (7 El. zw. 1 und 20)

```
g =

    1.0000    4.1667    7.3333   10.5000   13.6667   16.8333   20.0000
```

Felder: Zugriff auf Elemente

- `x=[1:3;4:6;7:9]`

```
x =

     1     2     3
     4     5     6
     7     8     9
```

- Mit Zeilen/Spaltenindex:
  `x(1,2)`

```
ans =

     2
```

- Bereiche (z.B. erste Zeile, Spalten 1 und 3):
  `x(1, [1 3])`

```
ans =

     1     3
```

- Bereich (: steht für ‘alle’)
  `x(1, :)`

```
ans =

     1     2     3
```

- Bereich (`end` steht für ‘letzter Index’)
  `x(1, [2:end])`

```
ans =

     2     3
```

Felder: Zugriff per linearem Index

- `x=[1:3;4:6;7:9]`

```
x =

     1     2     3
     4     5     6
     7     8     9
```

- Mit linearem Index:
  `x(4)`

```
ans =

     2
```

- Linearer Index:
  - Matrix wird auf eine Dimension reduziert
  - Reihenfolge: 1. Spalte, 2. Spalte, 3. Spalte etc.
  - Elemente werden durchnummeriert

Felder: logisches Indizieren

- `z=rand(3,3)`

```
z =

    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157
```

- Array mit Wahrheitswerten:
  `z < 0.5`

```
ans =

  3×3 logical array

   0   0   1
   1   1   1
   0   0   0
```

- Indizierung mit log. Array:
  `z( z < 0.5 )`

```
ans =

    0.1576
    0.4854
    0.1419
    0.4218
```

- Z.B. als Zuweisung
  `z( z < 0.5 ) = 0`

```
z =

    0.9649    0.9572         0
         0         0         0
    0.9706    0.8003    0.9157
```

Felder: Suchen in Arrays

- `z=rand(3,3)`

```
z =

    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157
```

- Liste von linearen Indizes:
  `find( z < 0.5 )`

```
ans =

     2
     5
     7
     8
```

- Zeilen/Spaltenindices:
  `[r,c] = find(z<0.5)`

```
r =     c =

     2       1
     2       2
     1       3
     2       3
```

Felder: Elemente ändern

- Alle Arten der Indizierung dürfen auch auf der linken Seite einer Zuweisung stehen
- `x(1,2) = 10`
- `x(:,2) = [11;12;13]`
- `x(:,2) = 42`

```
x =

     1     2     3
     4     5     6
     7     8     9
```

```
x =

     1     2     3
    10     5     6
     7     8     9
```

```
x =

     1    11     3
    10    12     6
     7    13     9
```

```
x =

     1    42     3
    10    42     6
     7    42     9
```

Felder: Elemente hinzufügen

- Hinzufügen: Element mit neuem Index zuweisen
  Matrix wird evtl. mit Nullen aufgefüllt

```
x =

     1     2     3
     4     5     6
     7     8     9
```

- `x(5,5) = 42`

```
x =

     1     2     3     0     0
     4     5     6     0     0
     7     8     9     0     0
     0     0     0     0     0
     0     0     0     0    42
```

- `x(6,:) = [1 2 3 2 1]`

```
x =

     1     2     3     0     0
     4     5     6     0     0
     7     8     9     0     0
     0     0     0     0     0
     0     0     0     0    42
     1     2     3     2     1
```

Felder: Elemente löschen

- Löschen: Elemente auf “leere Matrix” (`[ ]`) setzen

```
x =

     1     2     3
     4     5     6
     7     8     9
```

- Zweite Spalte löschen:
  `x(:,2) = [11;12;13]`

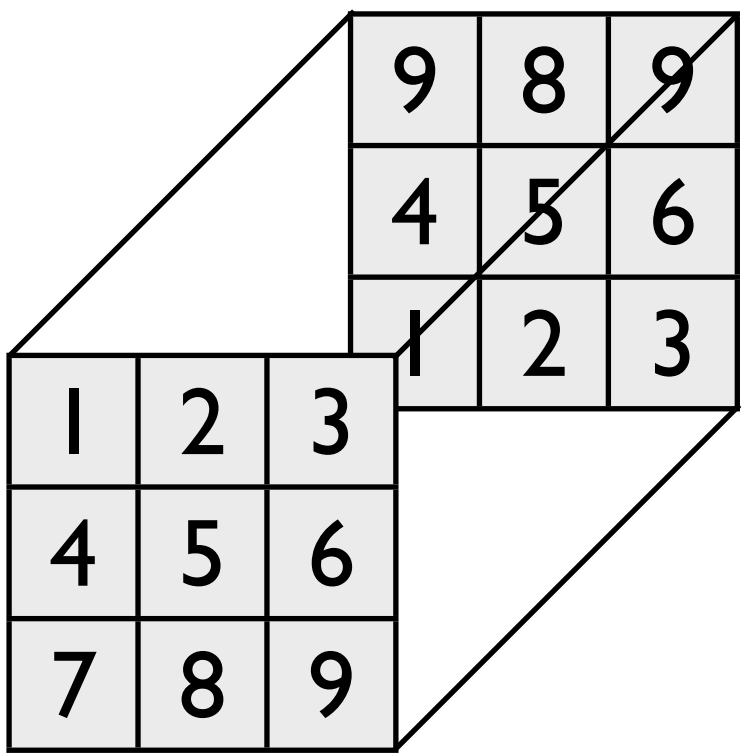
```
x =

     1     3
     4     6
     7     9
```

- Zeilen 1 und 3 löschen:
  `x([1 3],:) = []`

```
x =

     4     6
```

Mehrdimensionale Felder (3D ...)

- werden erzeugt, indem die dritte Dimension (und jede weitere) hinzugefügt wird



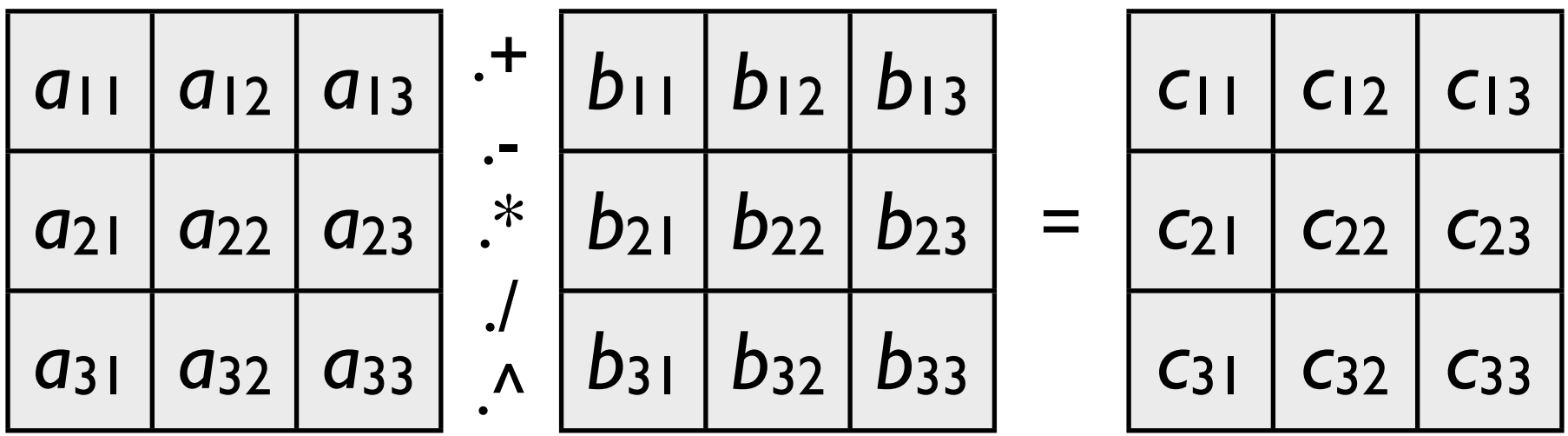

```
>> x=[1:3;4:6;7:9]

x =

     1     2     3
     4     5     6
     7     8     9

>> x(:,:,2)=[7:9;4:6;1:3]

x(:,:,1) =

     1     2     3
     4     5     6
     7     8     9


x(:,:,2) =

     7     8     9
     4     5     6
     1     2     3
```

Felder: Operationen

- Matlab unterscheidet:
  - Array-Operationen
    - elementweise
    - Felder müssen bgzl. Größe ‘kompatibel’ sein
    - beginnen mit . (z.B. **.***)
    - auch auf mehrdimensionale Felder anwendbar
  - Matrix-Operationen
    - aus linearer Algebra

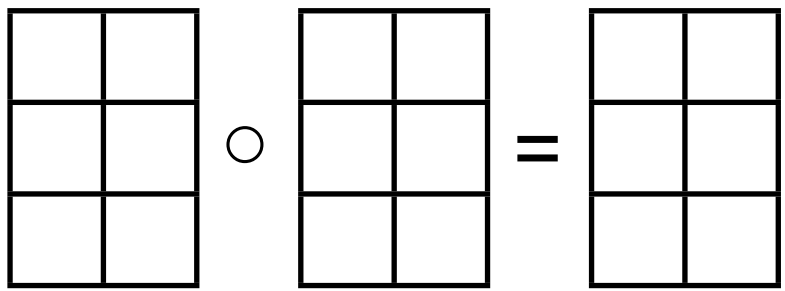
Array Operationen

- Unäres - : `c = -a` (d.h. $c_{ij} = -a_{ij}$)
- Transposition `':c = a'`



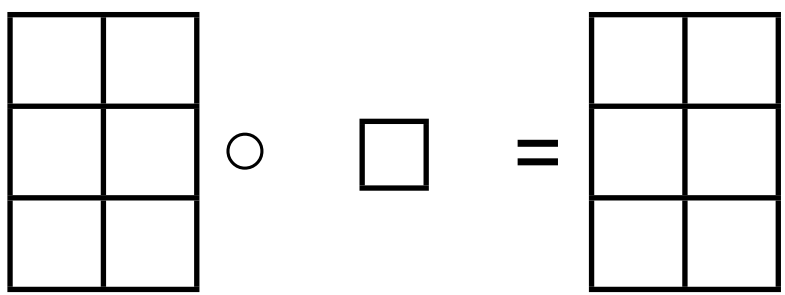
mit $c_{ij} = a_{ij}$.+ .- .* ./ .^ b_{ij}

Array Op.: Kompatible Größen



n x m n x m n x m

$c_{ij} = a_{ij} \circ b_{ij}$

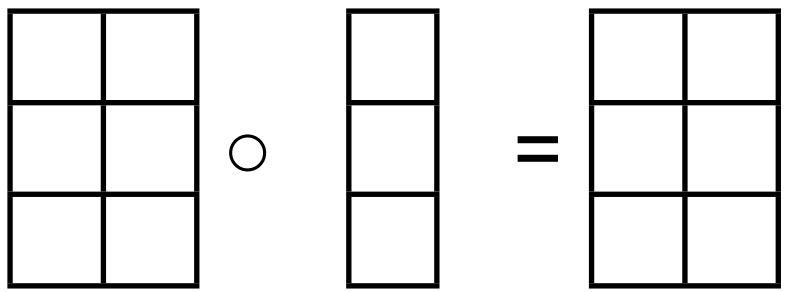


n x m 1 x 1 n x m

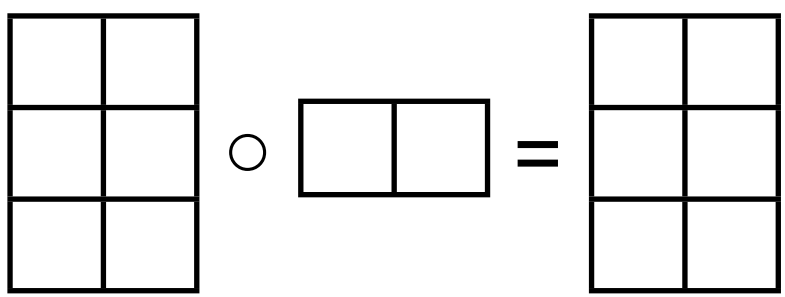
$c_{ij} = a_{ij} \circ b$

○: Operator

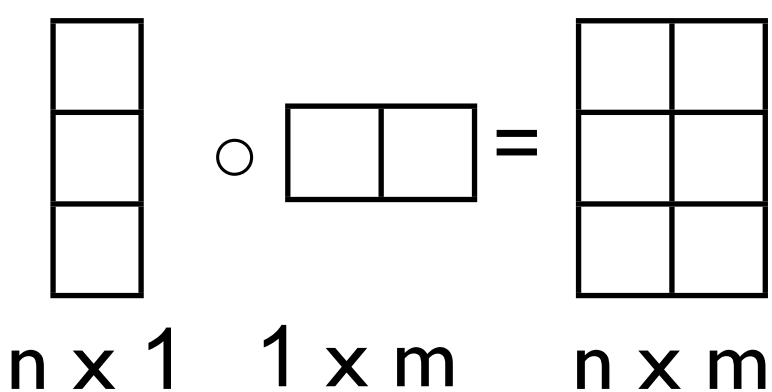
Array Op.: Kompatible Größen



$c_{ij} = a_{ij} \circ b_i$



$c_{ij} = a_{ij} \circ b_j$



$c_{ij} = a_i \circ b_j$

Matrix Operationen

- `C=A*B` Matrixmultiplikation $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$
- `x=A\B` Lösung des Gleichungsystems *A x* = *B*
- `x=B/A` Lösung des Gleichungsystems *x A* = *B*
- `C=A^B` Potenzierung (A oder B skalar)

Beispiel: Max. Element in Matrix

```
zz = randi(100, 1, 100);
fprintf('max. Element ist: %d\n',  maxi(zz));
fprintf('(mit Matlab Fkt: %d)\n',  max(zz));

function m = maxi( v )
    max_so_far = 0;
    for x = v
        if max_so_far < x
            max_so_far = x;
        end
    end
    m = max_so_far;
end
```

```
>> max_script
max. Element ist: 67
(mit Matlab Fkt: 67)
>>
```

Beispiel: Max. Element mit Index

```
zz = randi(100, 1, 10);
[e,x] = maxi(zz));
fprintf('max. Element ist: %d an %d\n',e,x);

function [m, max_idx] = maxi( v )
    max_so_far = v(1);
    for idx = [2:length(v)]
        if max_so_far < v(idx)
            max_so_far = v(idx);
            max_idx = idx;
        end
    end
    m = max_so_far;
end
```

```
>> max_script
max. Element ist: 67
(mit Matlab Fkt: 67)
>>
```