Anwendung von (Mathematica und) Matlab in der Physikalischen Chemie

3. Grundlegende Sprachkonzepte

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2015/16

Übersicht

UNI FREIBURG

Grundlegende Sprachkonzepte

- Syntax
- Datentypen
- Entscheidungsstrukturen
- Schleifen

Befehle/Funktionen

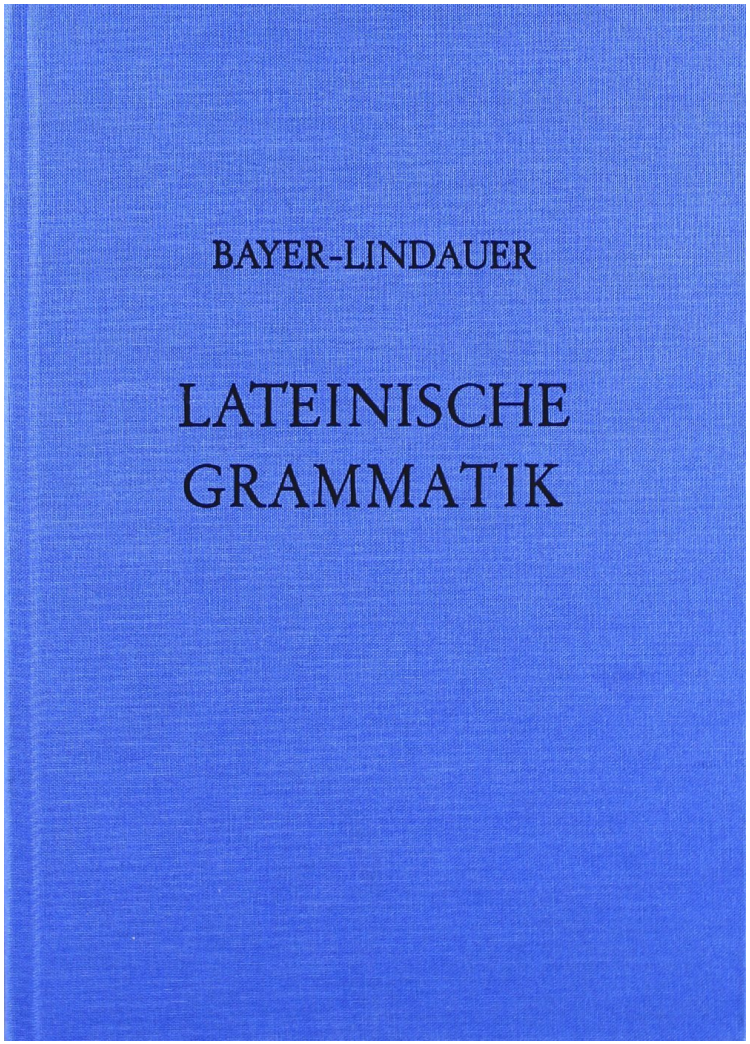
- Schnittstellen von Funktionen
- Kontext von Variablen

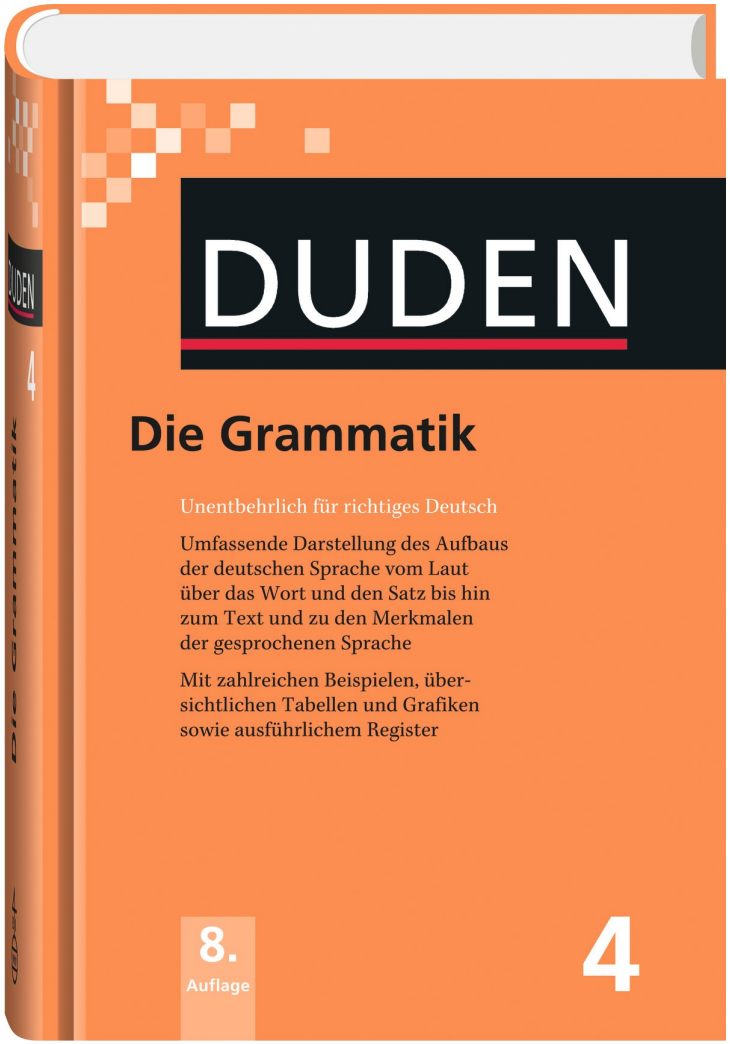
Hilfe zur Selbsthilfe

- Quellen für Hilfe
- Ein Wort zu „Google-Lösungen“

Grundlegende Sprachkonzepte

Grammatik und Syntax





Grammatik

- Datentypen
- Entscheidungsstrukturen
- Schleifen

Syntax

- Kommentare
- Zeilenenden
- Groß- und Kleinschreibung
- Zeilenumbrüche und Leerzeilen
- Einrückungen

Syntax

- Kommentare
  - Werden durch Prozentzeichen (%) eingeleitet
  - Alles nach dem „%“ in einer Zeile wird ignoriert.
- Zeilenenden
  - Befehle normalerweise mit Semikolon (;) beenden
  - Ansonsten wird der Variableninhalt ausgegeben
- Groß- und Kleinschreibung
  - Matlab unterscheidet zwischen Groß- und Kleinschreibung
- Zeilenumbrüche und Leerzeilen
  - Zeilenumbrüche in einem Befehl mit „. . .“
  - Leerzeilen ansonsten beliebig

Syntax

Listing 1: Beispiele für die grundlegende Syntax in Matlab

```
1  % Das ist ein Kommentar
2
3  sin(2*pi) % Beispiel fuer einen Kommentar nach einem Befehl
4
5  % Hier wird das Ergebnis ausgegeben
6  sin(2*pi)
7
8  % Hier wird nichts ausgegeben
9  sin(2*pi);
10
11 % Diese beiden Befehle sind _nicht_ identisch
12 sin(2*pi);
13 Sin(2*pi);
14
15 % Zeilenumbruch innerhalb eines Befehls
16 x = [ ...
17       1 2 3 ; ...
18       2 3 4 ; ...
19       3 4 5 ...
20     ];
```

UNI FREIBURG

Typisierung

Zuweisung eines Objekts einer Programmiersprache (zum Beispiel einer Variable) zu einem Datentyp

- Datentypen
  - Numerisch
  - Zeichen und Zeichenketten (*strings*)
  - Boolesche Ausdrücke
  - Komplexe Datentypen
- Bedeutung unterschiedlicher Datentypen
  - Mit Zeichenketten kann man (meist) nicht rechnen.
  - Befehle erwarten meist bestimmte Datentypen.

Numerische Datentypen

- Zwei Unterscheidungsmöglichkeiten
  1. Dimension
  2. Präzision/Wertebereich

Dimension

- Skalar (1×1)
- Vektor ($1 \times n$, $n \times 1$)
- Matrix ($n \times m$)

Präzision/Wertebereich

- (signed) integer
- real/float/double

☛ Matlab rechnet normal mit Gleitkommazahlen (double).

Numerische Datentypen

Listing 2: Numerische Datentypen unterschiedlicher Dimension in Matlab

```
1  % Skalar
2  number = 1;
3  emptyScalar = [];
4
5  % Vektoren
6  rowVector    = [1 2 3 4 5];
7  rowVector    = [1, 2, 3, 4, 5];
8  rowVector    = 1:5;
9
10 columnVector = [1; 2; 3; 4; 5];
11 columnVector = rowVector';
12
13 % Matrix
14 matrix = [ 1 2 3 ; 2 3 4 ; 3 4 5 ; 4 5 6 ];
```

- Zeilenvektor: eine Zeile/Reihe ($1 \times n$)
- Spaltenvektor: eine Spalte ($n \times 1$)

Indizierung numerischer Datentypen

- Gilt strenggenommen für alle geordneten Listen

  Geordnete Liste Struktur, deren Felder über einen ganzzahligen numerischen Index adressiert werden.

- Indices in Matlab immer in runden Klammern
  - Reihenfolge bei zweidimensionalen Matrizen: Reihe, Spalte
- Spezielle Indices
  - `end` – das letzte Element einer geordneten Liste
  - `:` – alle Elemente einer Dimension
- Zugriff auf Bereiche einer Dimension
  - Über eine Liste oder einen Bereich von Indices

Indizierung numerischer Datentypen

Listing 3: Indizierung geordneter Listen in Matlab

```
1  % Matrix
2  matrix = [ 1 2 3 ; 2 3 4 ; 3 4 5 ; 4 5 6 ];
3
4  % Erste Reihe der Matrix
5  firstRow = matrix(1,:);
6
7  % Zweite Spalte der Matrix
8  firstRow = matrix(:,2);
9
10 % Zweite und dritte Reihe der dritten Spalte
11 selection = matrix(2:3,3);
12 selection = matrix([2,3],3);
13
14 % Letzte Reihe der Matrix
15 lastRow = matrix(end,:);
16
17 % Erste bis vorletzte Spalte der Matrix
18 selection = matrix(:,1:end-1);
```

Zeichen und Zeichenketten

- `character`, `char`
  - Einzelnes Zeichen
- `string`
  - Zeichenkette aus einzelnen Zeichen (`char`)
- Zeichenketten in Matlab
  - Zeichenketten immer in Hochkommata ('...') eingeschlossen
  - Mehrdimensionale Strings (Array von Zeichenketten): Reihen müssen gleiche Spaltenzahl haben

☛ Mehrzeilige Texte in `cell arrays` ablegen

Boolesche Ausdrücke

- Zwei Werte
  - wahr (*true*), unwahr (*false*)
- Matlab
  - `true`, `false`
  - `0` gilt als `false`
  - $\neq$`0` gilt als `true`

George Boole
(1815–1864)

Mathematical Association of America

Komplexe Datentypen

- `cell array` (Liste)
  - Daten unterschiedlicher Typen und Größen
  - In den Feldern eines Datenfeldes (*array*) gespeichert
  - „Generalisiertes“ Datenfeld (*array*)
  - Felder numerisch (mit ganzen Zahlen) indiziert
- `structure` (Wörterbuch)
  - Daten unterschiedlicher Typen und Größen
  - In den Feldern einer Struktur gespeichert
  - Assoziatives Datenfeld
  - Felder mit Namen (*strings*) indiziert

☛ Beide sind hierarchisch verschachtelbar.

Komplexe Datentypen

Geordnete Listen

#	Wert
1	0.0000
2	0.0025
3	0.0050
⋮	
n-1	0.2475
n	0.2500

#	Wert
1	'Im'
2	'Anfang'
3	'war'
⋮	
n-1	'die'
n	'Tat'

Assoziative Datenfelder

Schlüssel	Wert
Name	'K. Racht'
Alter	42
Adresse	
Hobbies	{'...','...'}

Schlüssel	Wert
Straße	'Talstraße'
Nummer	21

cell arrays

Listing 4: cell arrays in Matlab

```
1  % Empty cell array
2  C = cell(0);
3
4  % Cell array of strings
5  C = {'Im','Anfang','war'};
6
7  % Cell array with different types
8  C = {'String',[1 2 3],'String',[1 2 3; 2 3 4; 3 4 5]};
9
10 % Same cell array as above
11 C{1} = 'String';
12 C{2} = [1 2 3];
13 C{3} = 'String';
14 C{4} = [1 2 3; 2 3 4; 3 4 5];
15
16 % Accessing a field
17 foo = C{1}; % Returns a string
18 foo = C(1); % Returns a 1x1 cell
```

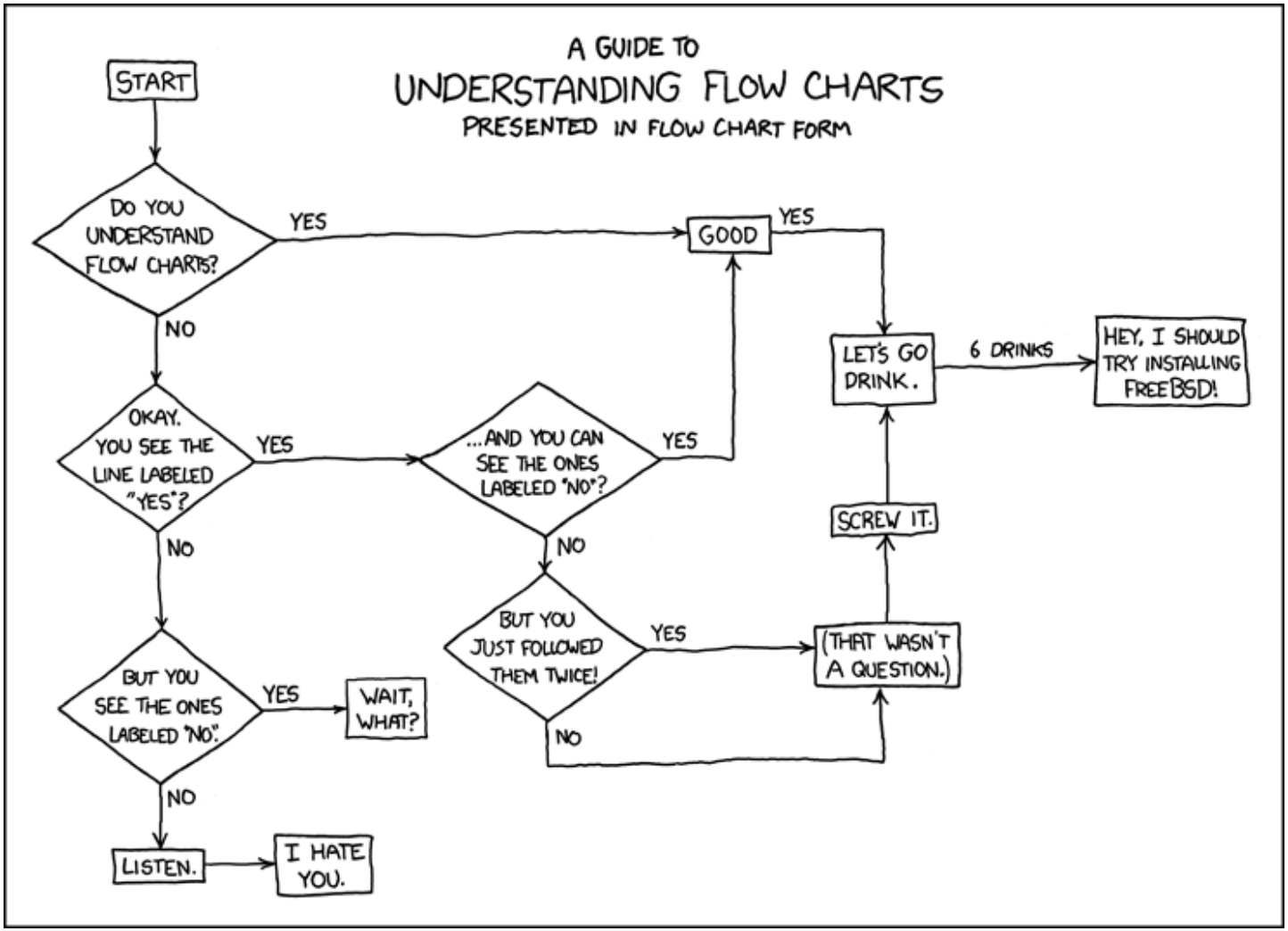
structures

Listing 5: structures in Matlab

```
1  % Empty structure
2  S = struct();
3
4  % Structure with some field and value
5  S.field = 'value'
6
7  % Structure with fields of different types
8  S.field1 = 'value';
9  S.field2 = pi;
10 S.field3 = [1 2 3];
11
12 % Same structure as above
13 S = struct(...
14     'field1','value',...
15     'field2',pi,...
16     'field3',[1 2 3] ...
17     );
18
19 % Accessing a field
20 foo = S.field3;
```

Grundlegende Sprachkonzepte

Entscheidungsstrukturen



https://xkcd.com/518/

Konditionale Strukturen

- `if...else`
  - Testet auf bestimmte Bedingung
  - Mehrere Bedingungen über logische Operatoren verknüpft

Logische Operatoren

- Arten logischer Operatoren
  - AND (`&`), OR (`|`), EQUAL (`eq()`, `==`), NOT (`not()`,`~`)
  - Klammern zur Gruppierung logischer Ausdrücke
- „Kurzschluss-Operatoren“
  - `&&`, `||`
  - Überprüfung bricht ab, sobald die Bedingung erfüllt ist
  - Beispiel: `A && B && C` bricht nach `A` ab, wenn `A` unwahr

Listing 6: Einfachste Form einer if-Struktur in Matlab

```
1 if <condition>
2     % do something
3 end
```

- Abbruch der weiteren Abarbeitung und Rückkehr zum Aufrufer über `return`

Listing 7: if-Struktur mit Alternativzweig

```
1 if <condition>
2     % do something
3 else
4     % do something else
5 end
```

☛ Tipp: Invertierte Logik spart häufig den `else`-Zweig

Entscheidungsstrukturen – Zwei praktische Beispiele

Listing 8: Reales Beispiel einer if-Struktur in Matlab

```
1  % Compare current year
2  if str2double(datestr(now,'yyyy')) < 2016
3      disp('You''re outdated.');
4  else
5      disp('You''re in time.');
6  end
7
8  % "now"        - returns current date and time
9  % "datestring" - formats date - here, "yyyy" means four-digit year only
10 % "str2double" - converts string into number for comparison
```

Listing 9: Überprüfung der Zahl der Eingabeparameter

```
1  % nargin returns the number of input arguments of a function
2  if nargin ~= 2
3      return;
4  end
```

Listing 10: if-Struktur mit mehreren Alternativbedingungen

```
1  if <condition1>
2      % do something
3  elseif <condition2>
4      % do something else
5  else
6      % do something else
7  end
```

Listing 11: if-Strukturen lassen sich verschachteln

```
1  if <condition1>
2      if <additionalCondition>
3          % do something
4      else
5          % do something else
6      end
7  elseif <condition2>
8      % do whatever
9  else
10     % give up
11 end
```

Fallunterscheidungen

- `switch...case`
- Vorteile gegenüber `if...elseif...else`
  - Oft übersichtlicher
  - Gut für Unterscheidung mehrerer Fälle (>2) geeignet
- Beschränkungen von Matlab
  - Nur Skalare oder Zeichenketten (*Strings*) als Schalter
  - Keine Bedingungen in den Fällen

Listing 12: Einfachste Form einer switch-case-Struktur in Matlab

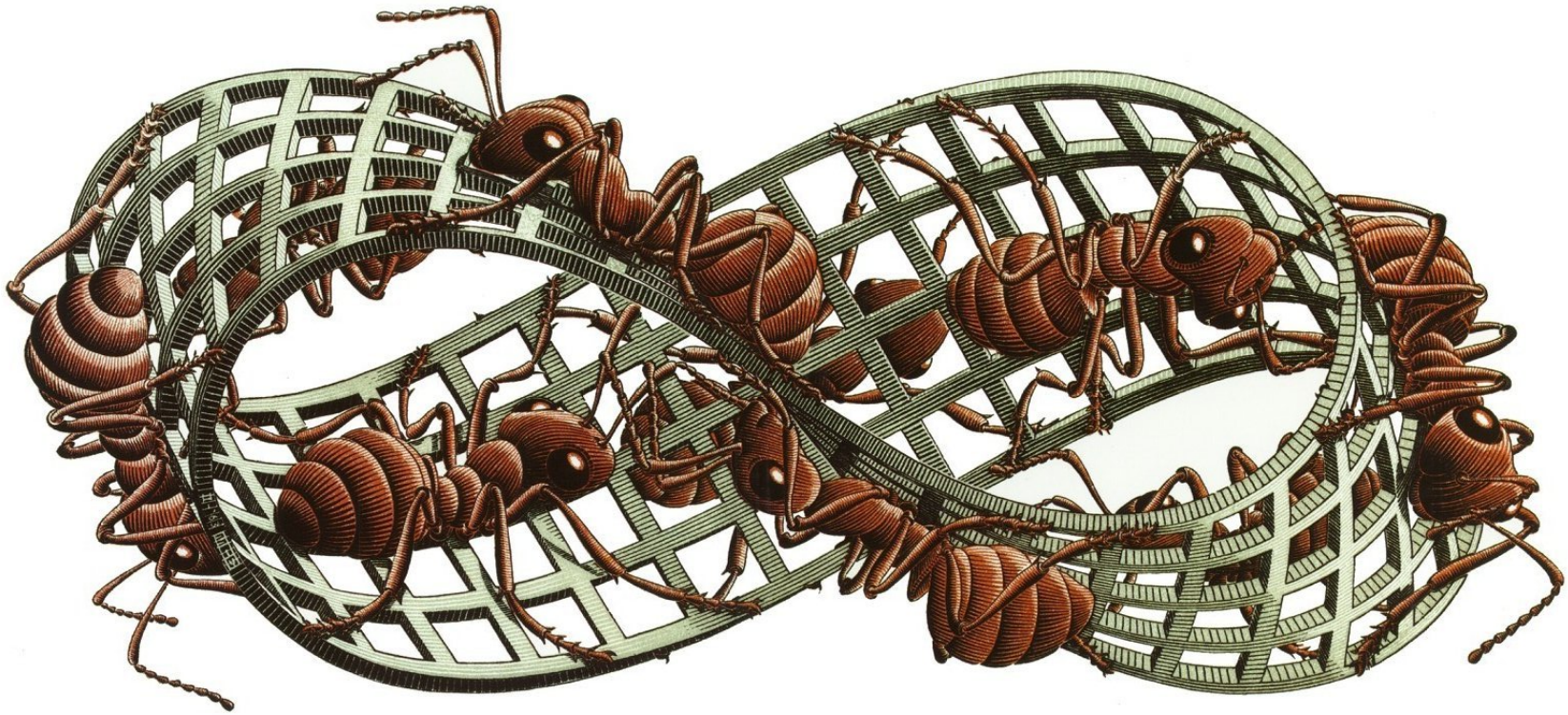
```
1 switch switch_expression
2     case case_expression
3         statements
4     case case_expression
5         statements
6     % ...
7 end
```

Listing 13: switch-case-Struktur mit otherwise-Zweig

```
1 switch switch_expression
2     case case_expression
3         statements
4     case case_expression
5         statements
6     % ...
7     otherwise
8        statements
9 end
```

Grundlegende Sprachkonzepte

Schleifen



M. C. Escher: Möbius-Band II

Schleifen

- `for`-Schleifen
  - Definierte Anzahl an Schleifendurchläufen
  - Typisches Einsatzgebiet:
    Iterieren über die Elemente eines Vektors
- `while`-Schleifen
  - Schleifendurchlauf, solange eine Bedingung wahr ist
  - Typisches Einsatzgebiet:
    Zeilenweises Einlesen einer Datei

☞ Abbruch einer Schleife über `break`

`for`-Schleifen

Listing 14: Einfachste Form einer for-Schleife in Matlab

```
1 for loopIndex = start : stop
2     % Do something
3 end
```

- `loopIndex` wird in jedem Durchlauf um 1 erhöht

Listing 15: for-Schleife mit angegebenem Inkrement

```
1 for loopIndex = start : increment : stop
2     % Do something
3 end
```

- `increment` kann negativ und nicht-ganzzahlig sein

`for`-Schleifen: Ein praktisches Beispiel

Listing 16: Iterieren über alle Elemente eines Vektors

```
1 % Define vector
2 x = 1:0.1:2*pi:
3
4 % Loop over each element of vector x
5 for k = 1 : length(x)
6     y = sin(x(k));
7 end
```

- Anmerkungen
  - Als Laufvariable *nie* `i` oder `j` verwenden (komplexe Zahl)
  - Als Laufvariable *nie* `l` verwenden (`1` oder `l`?)
- Eigenheiten von Matlab
  - `for`-Schleifen sind langsam, lassen sich oft vermeiden

`while`-Schleifen

Listing 17: Einfachste Form einer while-Schleife in Matlab

```
1 while condition
2     % Do something
3 end
```

- Bedingung wird am Anfang jedes Durchlaufs überprüft.
  - Matlab kennt (anders als andere Sprachen) keine Schleifen, die die Bedingung erst am Ende überprüfen.
- Bedingung muss sich innerhalb der Schleife ändern.
  - Wird sonst zur „Endlosschleife“

Befehle/Funktionen

Die „Wörter“ einer Programmiersprache

Befehle/Funktionen

Die „Wörter“ einer Programmiersprache

UNI FREIBURG

Befehle und Funktionen

- Befehle sind Funktionen
- (Selbst geschriebene) Funktionen sind Befehle

Möglichkeiten des Aufrufs

1. Kommandozeile
   - Befehl wird direkt eingetippt und ausgeführt
2. Skript
   - Liste von Befehlen in einer Datei
3. Funktion selbst schreiben
   - Grund: „Ich will etwas tun, was Matlab nicht kann.“
   - Spracherweiterung mithilfe vorhandener Befehle

Funktionen in Matlab

- Funktionsname und Dateiname müssen übereinstimmen.
- Nur eine Funktion pro Datei (Ausnahme: Unterfunktionen)

Benennung von Funktionen in Matlab

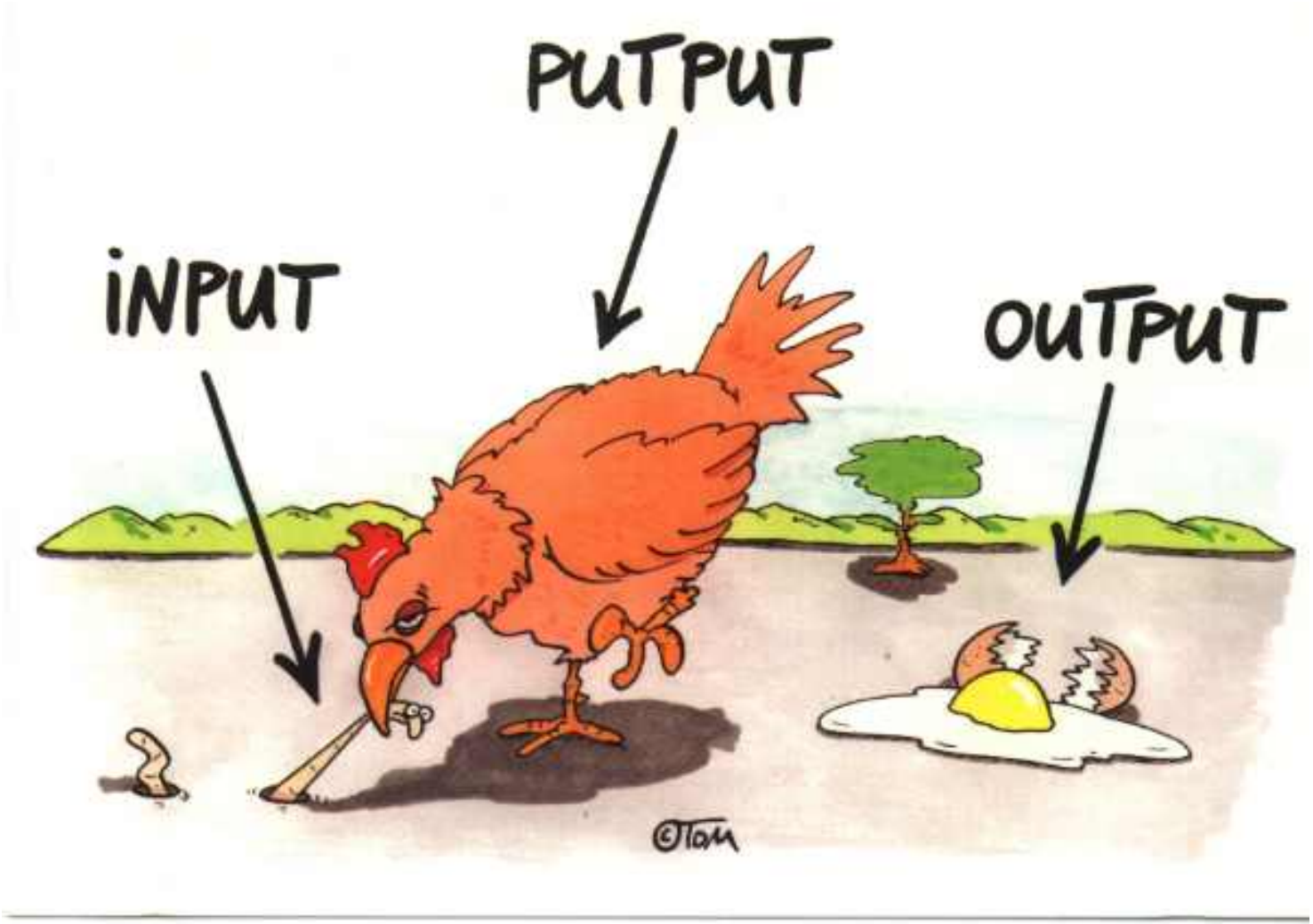
- Matlab unterscheidet zwischen Groß- und Kleinschreibung.
- Funktionsnamen müssen mit einem Buchstaben beginnen.
- Sonderzeichen sind nicht erlaubt. (Ausnahme: „_“)

☛ Tipp: Sprechende Namen erhöhen die Lesbarkeit

☛ Tipp: Präfix zur Vermeidung von Doppelungen

Befehle/Funktionen

Klar definierte Schnittstellen



Thomas Körner, alias ©TOM

Listing 18: Funktionsdeklaration in Matlab

```
function [out1,out2] = myFunction(in1,in2)
```

Funktionsdeklaration in Matlab

1. Schlüsselwort „`function`“
2. Liste der Rückgabeparameter (output)
3. Funktionsname
4. Liste der Übergabeparameter (input)

☛ Es gibt Funktionen *ohne* Parameter

Möglicher Kontext einer Variablen

lokal nur für die jeweilige Funktion „sichtbar“

global für alle Funktionen „sichtbar“

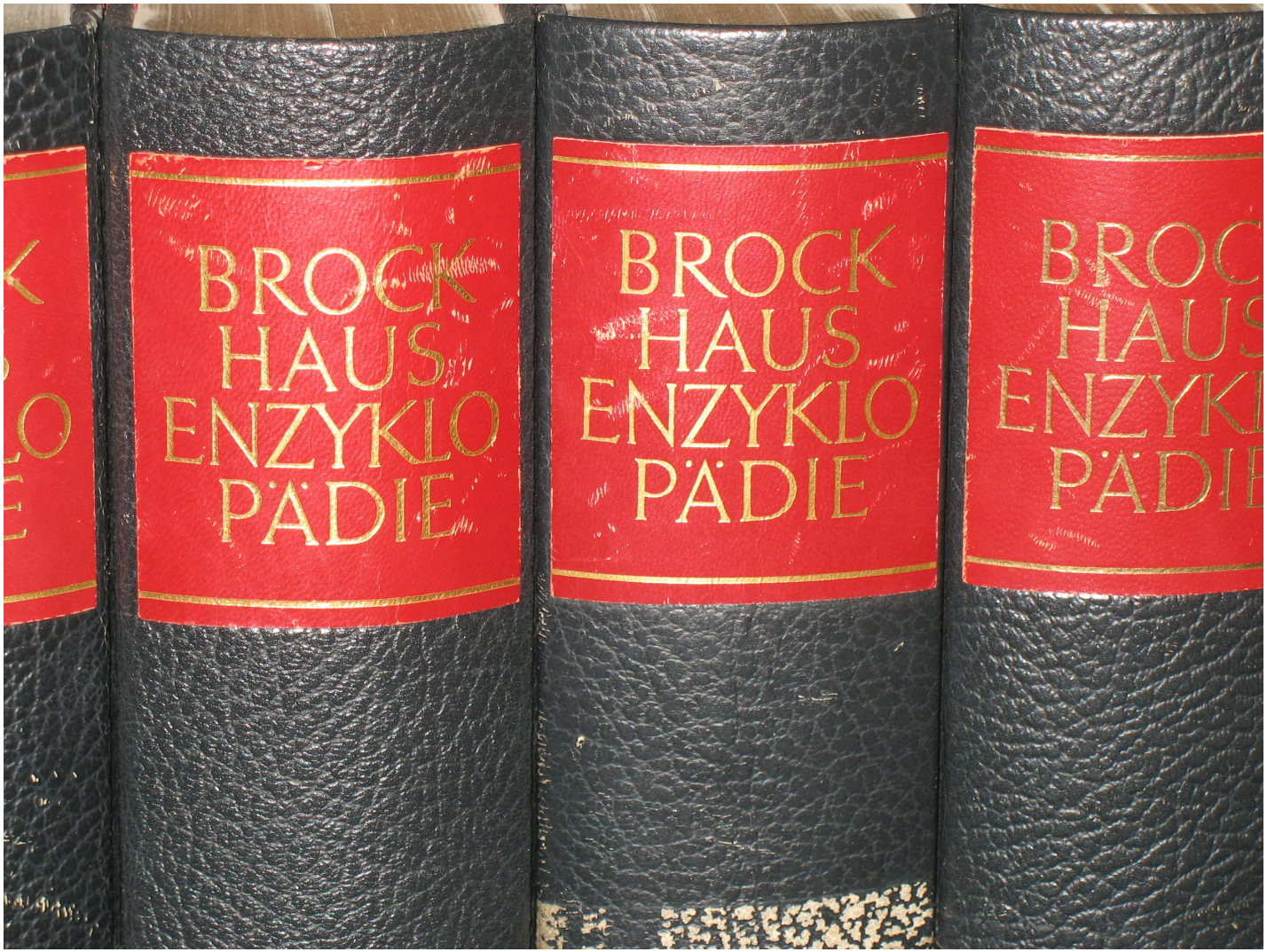
Konsequenzen

- Unterschiedliche Funktionen können Variablen mit dem gleichen Namen verwenden.
- Eine Funktion kennt nur globale, ihr übergebene oder in ihr definierte Variablen.

☞ Skripte haben (im Gegensatz zu Funktionen) Zugriff auf alle Variablen im Matlab-„Workspace“.

Hilfe zur Selbsthilfe

Dokumentation zur Hand haben und nutzen



Satz

Man muss nicht alles wissen, sollte aber wissen, wo es steht.

- Programmieren lernen ist wie eine Sprache lernen.
- Grundlegende Sprachkonzepte müssen bekannt sein.
- Details können in der Dokumentation nachgeschlagen werden.

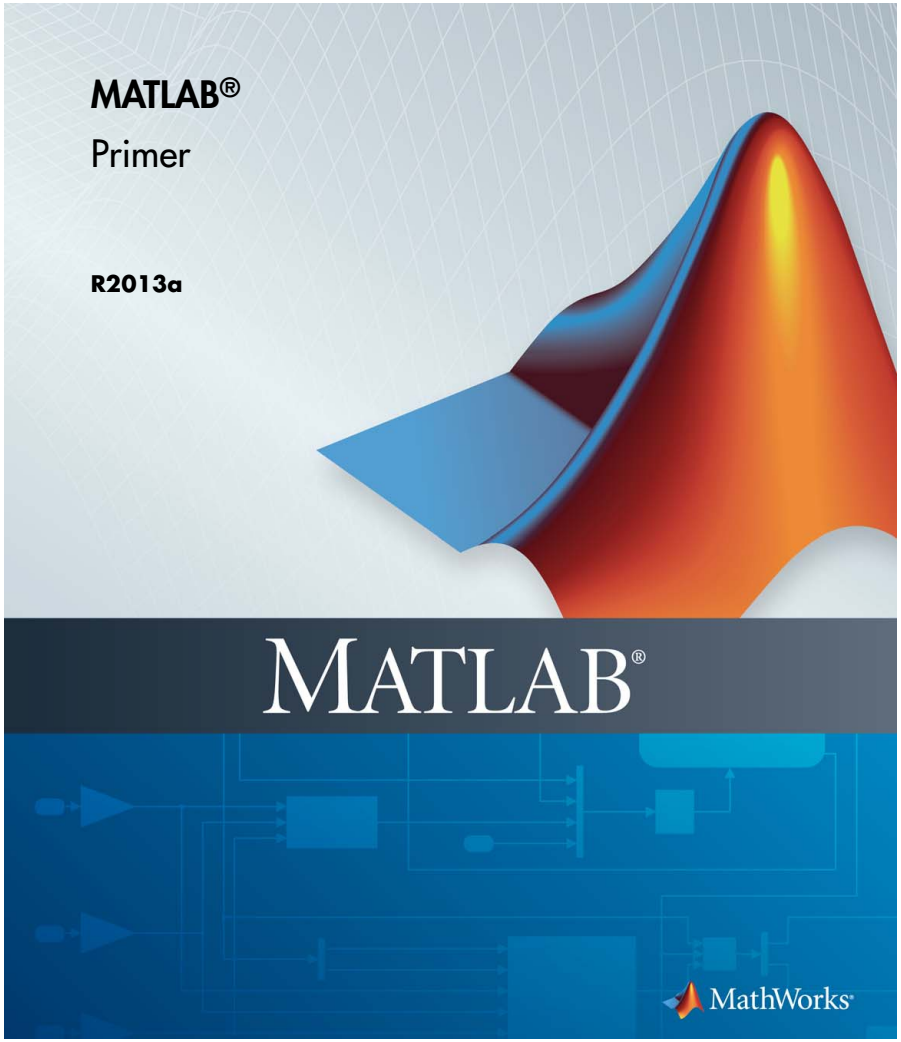
☞ Kenntnis der vorhandenen Dokumentation und wie man sie nutzt.

Offline verfügbar

- Eingebaute Hilfe in Matlab
  - `help <Befehlsname>` (auf der Kommandozeile)
  - `doc <Befehlsname>` (eigenes Fenster, ausführlicher)
- Handbücher zu Matlab und kommerziellen Toolboxen
  - mittlerweile nur noch elektronisch als PDF-Dokumente
  - Zugriff nur mit Konto bei der MathWorks-Seite
- Bücher
- Kollegen, Betreuer, Freunde

Hilfe zur Selbsthilfe

Hilfe und Dokumentation



Inhalte

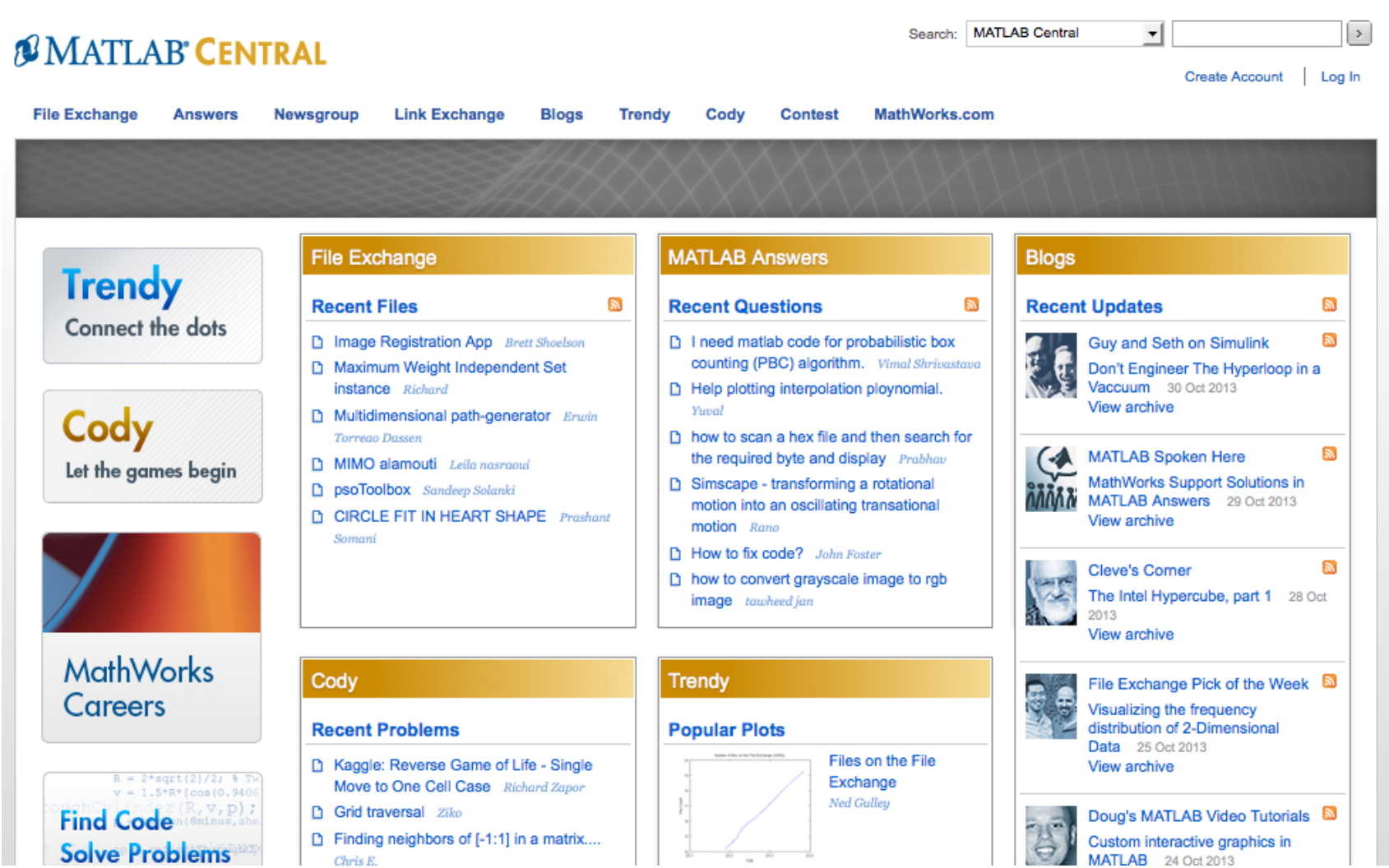
- Quick Start
- Language Fundamentals
- Mathematics
- Graphics
- Programming

Online verfügbar

- MathWorks-Webseite
  - Webcasts (kleine Filme) zur Einführung
  - Matlab Central
  - Matlab File Exchange
- `http://undocumentedmatlab.com/`
  - Richtet sich eher an Experten
  - Sehr viele Interna zu Matlab
- (manche) Kollegen, Betreuer, Freunde
- `http://lmgtfy.com/`

Hilfe zur Selbsthilfe

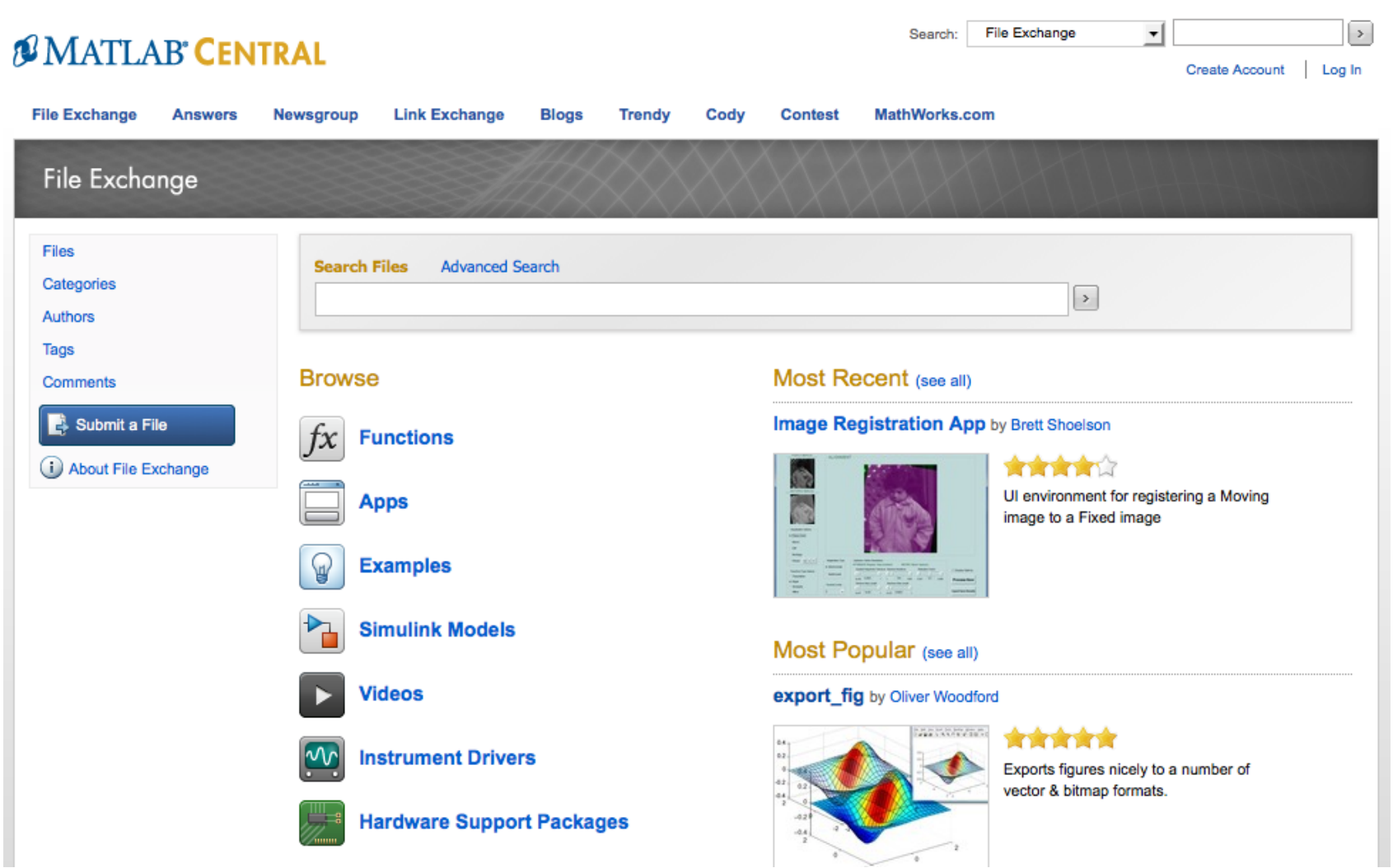
Hilfe und Dokumentation: Matlab Central



http://www.mathworks.com/matlabcentral/

Hilfe zur Selbsthilfe

Hilfe und Dokumentation: Matlab File Exchange



`http://www.mathworks.com/matlabcentral/fileexchange/`

Ein Wort zu „Google-Lösungen“

- Code immer erst verstehen und dann einsetzen
- Schwarm-Intelligenz sorgt meist nicht für bessere Code-Qualität.
- Google verhilft zu schnellen Lösungen – aber: Oft sind „offizielle“ oder spezifische Quellen besser.

☛ Viele Wege führen nach Rom.
Man kann von anderen viel lernen, sollte sich aber immer die Mühe machen, deren Code zu verstehen.

Ausblick

Wie geht's weiter?

UNI FREIBURG

...gleich geht's weiter

Vorschau: Schritte in die Praxis

- Daten importieren
- Daten verarbeiten
- Abbildungen: Daten darstellen