Karlsruher Institut für Technologie

MINT-KOLLEG BADEN-WÜRTTEMBERG

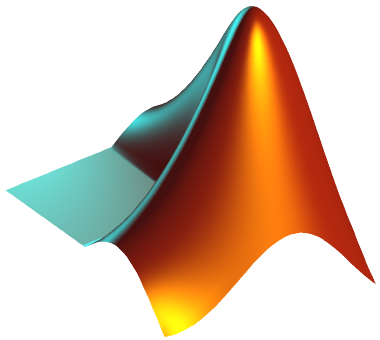
Einfach Programmieren lernen mit MATLAB!

Dr. V. Rutka, Dr. J. Liedtke
E-Mail: vita.rutka@kit.edu, juergen.liedtke@kit.edu

MINT-Kolleg Baden-Württemberg

MATLAB = MATrix LABoratory

- entwickelt von Cleve Moler, Ende der 1970er Jahre, Universität New Mexico
- vielfältige Visualisierungsmöglichkeiten
- komfortable Entwicklungsumgebung
- umfangreiche Erweiterungen (Toolboxen, Simulink,. . .)
- (vergleichsweise) einfach zu erlernende Interpreter-Sprache
- ausführliche Dokumentation (auf Englisch), viele Lehrbücher auch auf Deutsch
- Dokumentation, Downloads, Videos, Blogs, Presse, Arduino, Raspberry Pi, Tutorials, . . . : siehe www.mathworks.de



Einfach Programmieren lernen mit MATLAB!

Karlsruher Institut für Technologie

1. Die ersten Schritte

Mein erstes MATLAB Programm

Mit Buchstaben rechnen: Variablen

2. Etwas Mathematik

Arithmetische Operatoren und mathematische Funktionen

Viele Zahlen auf einmal: Vektoren in MATLAB

Visualisierung geometrischer Figuren

3. Ablaufstrukturen I

„Richtiges“ Programmieren: Ablaufstrukturen

Verzweigungen

4. Ablaufstrukturen II

Schleifen (while-Schleife)

Schleifen und Verzweigungen kombinieren

Karlsruher Institut für Technologie

MINT-KOLLEG BADEN-WÜRTTEMBERG

Einfach Programmieren lernen mit MATLAB!

1. Die ersten Schritte

MINT-Kolleg Baden-Württemberg

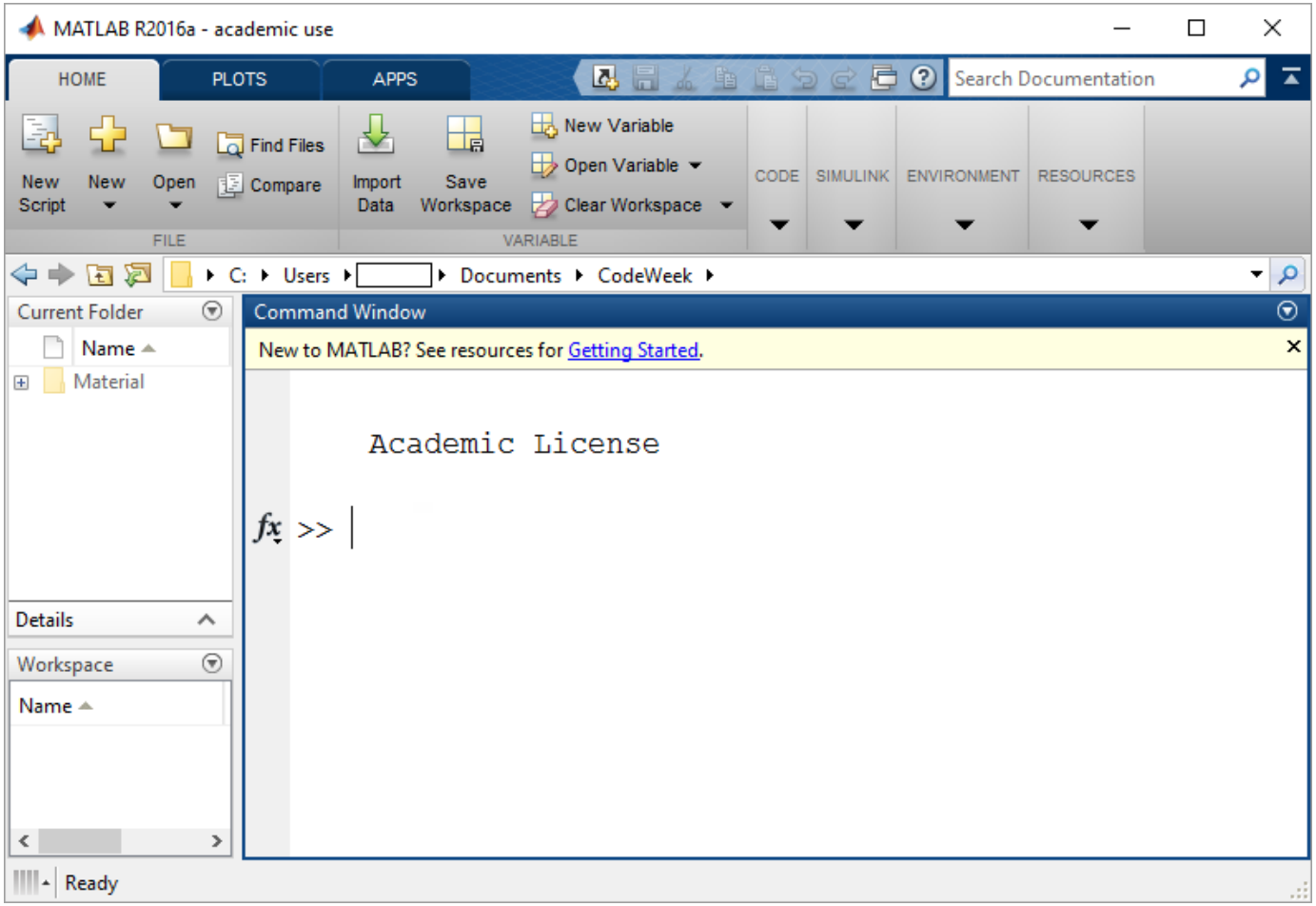
1. Die ersten Schritte

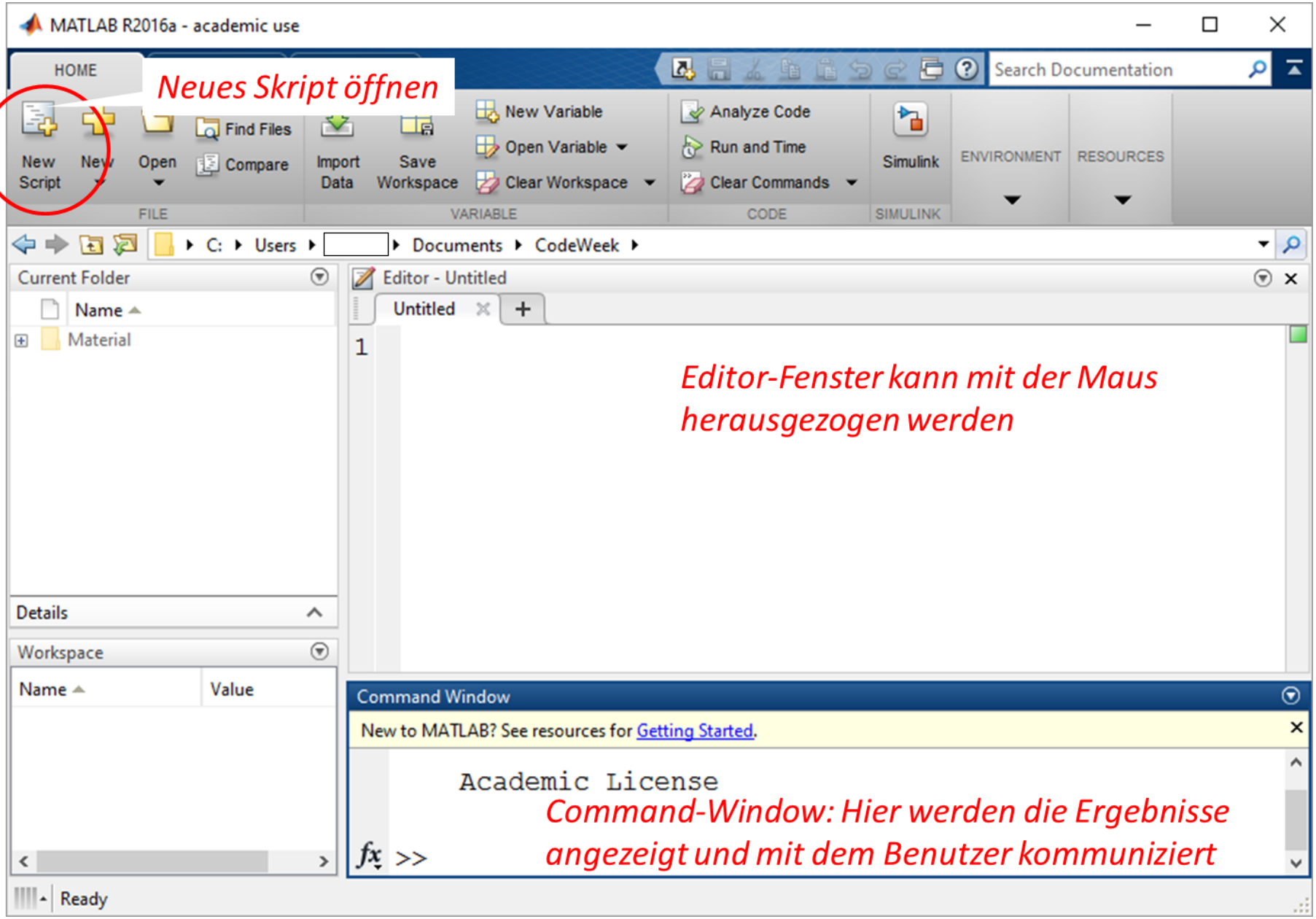
Karlsruher Institut für Technologie

Die grafische Benutzeroberfläche von MATLAB

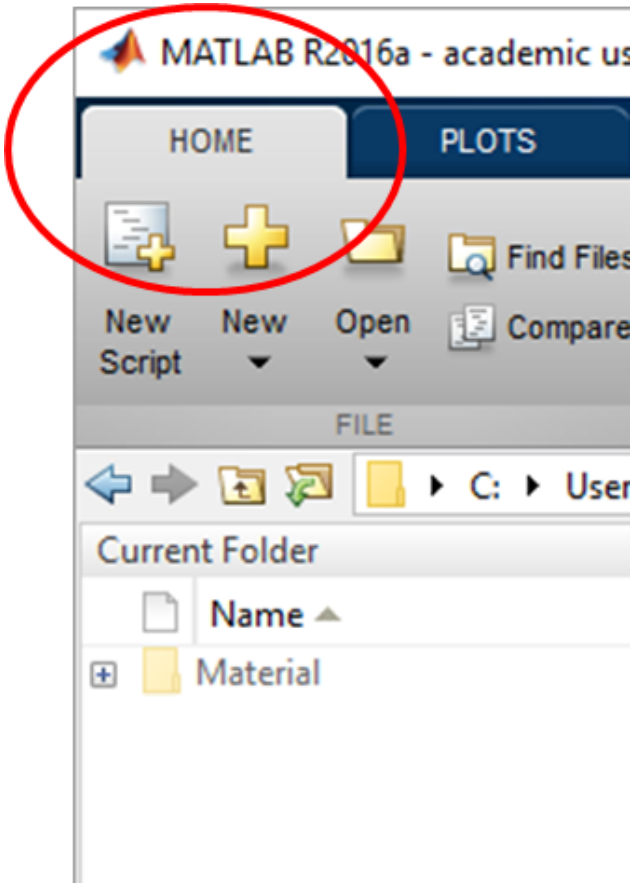
- Doppelklick auf das MATLAB-Icon (Windows)

- Eingabe von matlab in der Kommandozeile (Linux)

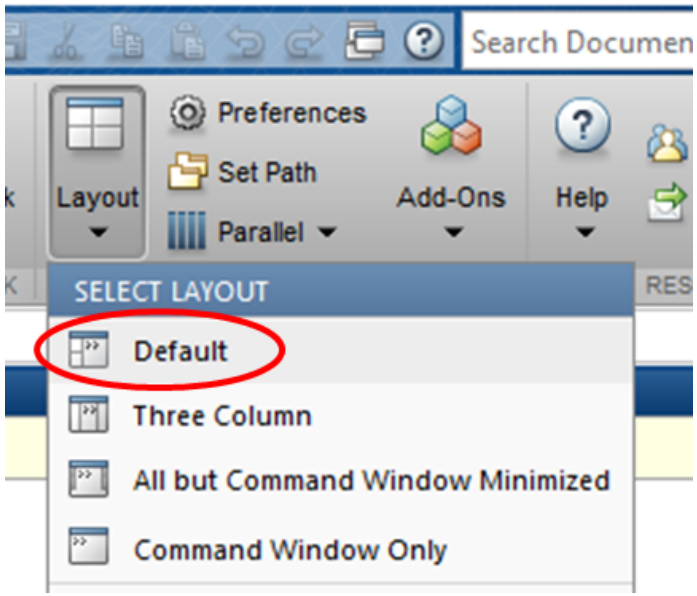



MATLAB R2016a - academic use
HOME
Neues Skript öffnen
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Simulink
ENVIRONMENT
RESOURCES
FILE
VARIABLE
CODE
SIMULINK
Search Documentation
C: ▸ Users ▸ Documents ▸ CodeWeek ▸
Current Folder
Name
Material
Details
Workspace
Name
Value
Editor - Untitled
Untitled
1
Editor-Fenster kann mit der Maus herausgezogen werden
Command Window
New to MATLAB? See resources for Getting Started.
Academic License
Command-Window: Hier werden die Ergebnisse angezeigt und mit dem Benutzer kommuniziert
>>
Ready

HILFE! Wie sieht jetzt mein MATLAB aus?!?



1. Schritt: Fenster ausreichend breit ziehen,
2. Schritt: zum Register *HOME* wechseln



3. Schritt: *Layout*-Knopf suchen
4. Schritt: *Default* auswählen

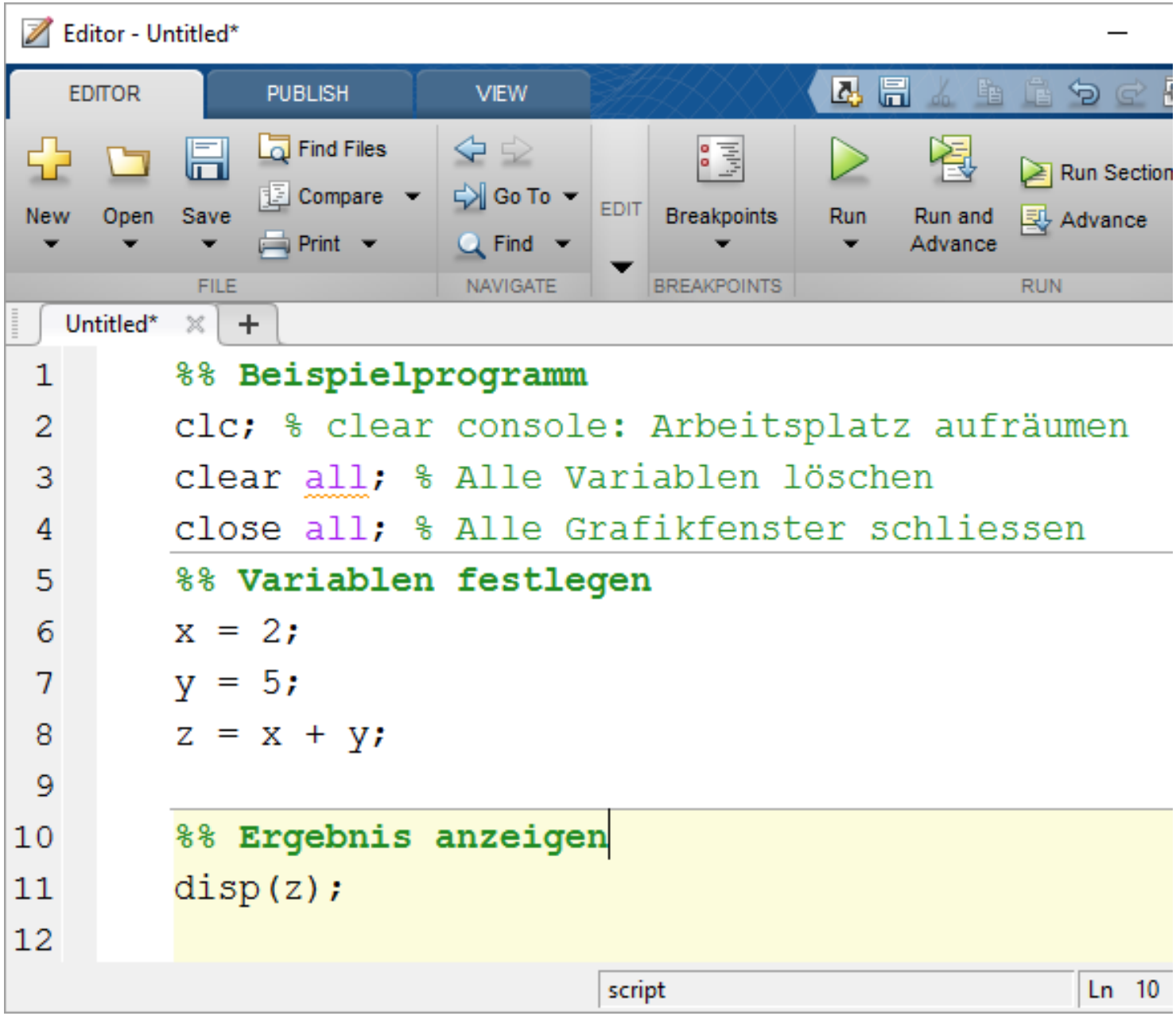
Die ersten Programme: Skripte

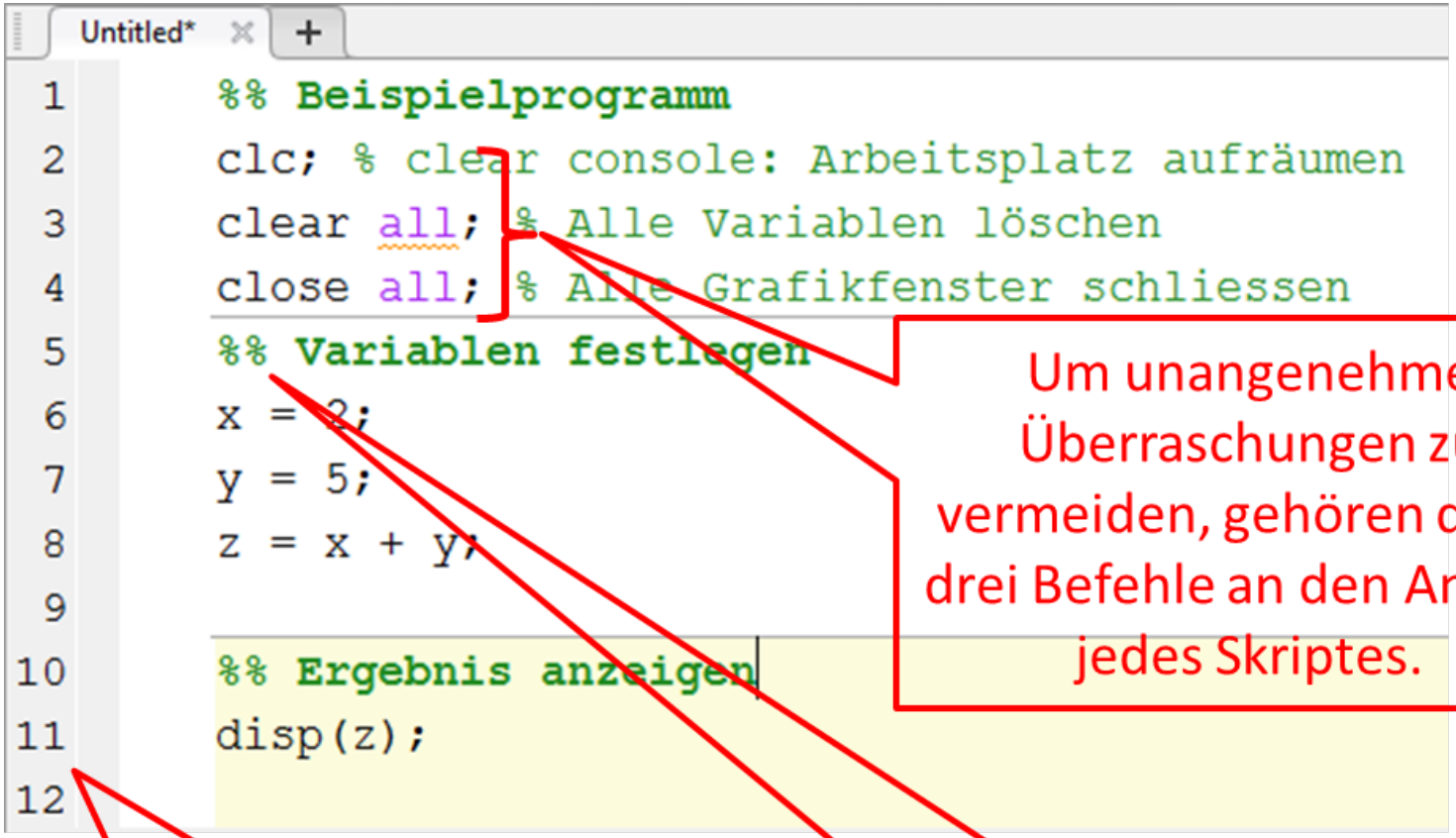
Zwei Typen von MATLAB-Programmen (**.m-Dateien*):

- *Skripte:*
  „einfaches“ Aneinanderreihen von MATLAB-Befehlen. Die Befehle werden von oben nach unten der Reihe nach ausgeführt.
- *Funktionen:*
  (Prozeduren und) Funktionen bezeichnen *Unterprogramme* zur Darstellung von Teilalgorithmen.

Prinzip ähnlich wie in den meisten Programmiersprachen.

Jetzt schreiben wir unser erstes MATLAB-Programm (ein *Skript*). Am besten ist es, den Schritten auf den nächsten Folien einfach zu folgen.


Editor - Untitled*
EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
RUN
Untitled*
1 %% Beispielprogramm
2 clc; % clear console: Arbeitsplatz aufräumen
3 clear all; % Alle Variablen löschen
4 close all; % Alle Grafikfenster schliessen
5 %% Variablen festlegen
6 x = 2;
7 y = 5;
8 z = x + y;
9
10 %% Ergebnis anzeigen
11 disp(z);
12
script
Ln 10

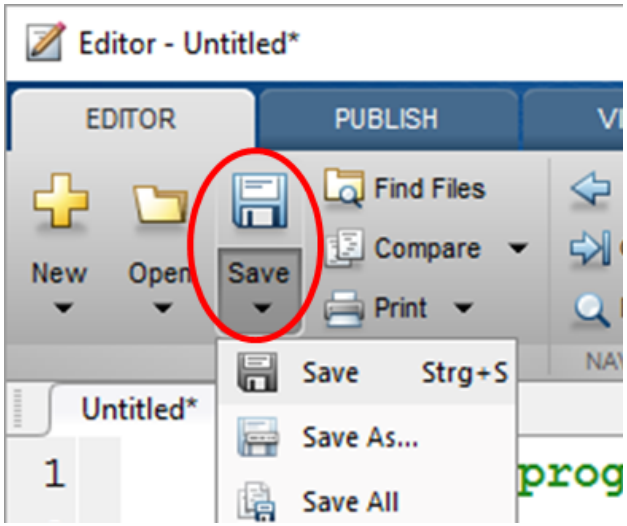


Um unangenehme Überraschungen zu vermeiden, gehören diese drei Befehle an den Anfang jedes Skriptes.

Zeilen werden automatisch nummeriert. Das erleichtert später die Fehlersuche.

Mittels eines doppelten %-Zeichens kann das Skript in Abschnitte geteilt werden.

Skripte speichern

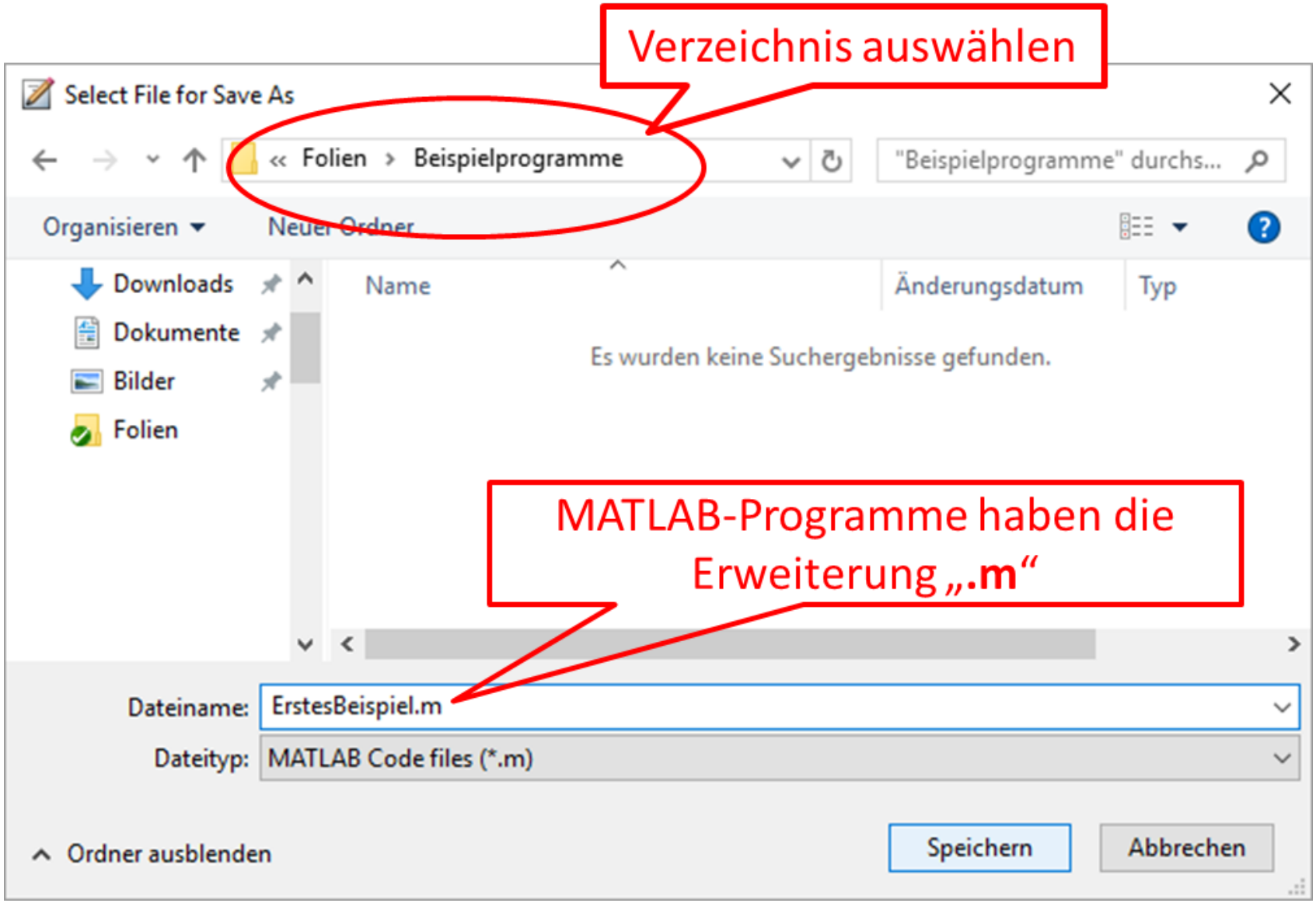


Zuerst muss das Programm gespeichert werden. Der *„Save“*-Knopf befindet sich im *EDITOR*-Register.

Regeln für Dateinamen:

- Der Name besteht aus Buchstaben und Ziffern, wobei die Zahlen NICHT direkt am Anfang stehen dürfen.
- Als Sonderzeichen sind NUR Unterstriche „_“ erlaubt.
  Nicht erlaubt sind z.B. Leerzeichen, arithmetische Operatoren (+,-,/,\,*,^), Punkt, Komma, Semikolon und andere Sonderzeichen (<,>,(,),{,},~,#,&,% usw.).
- Erweiterung ist „.m“.

Skripte speichern



Dateinamen

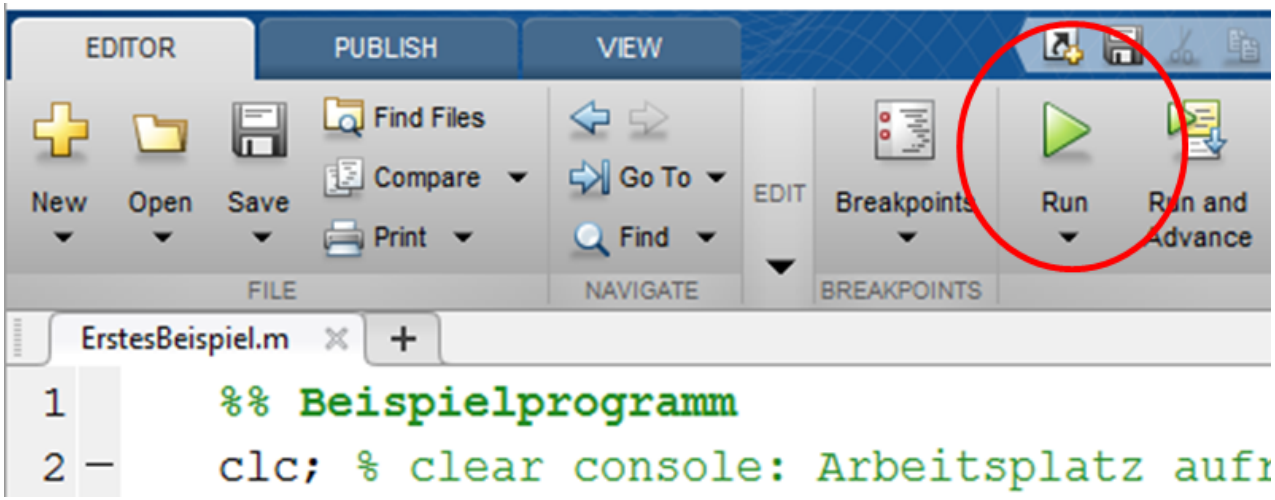
Geeignet	Fehlerhaft
ErstesBeispiel.m	Erstes Beispiel.m
Erstes_Beispiel.m	Erstes-Beispiel.m
Beispiel1.m	1Beispiel.m
	Beispiel1.m.m

Achtung:
Die vorhandenen MATLAB-Befehle sollten nicht überschrieben werden!

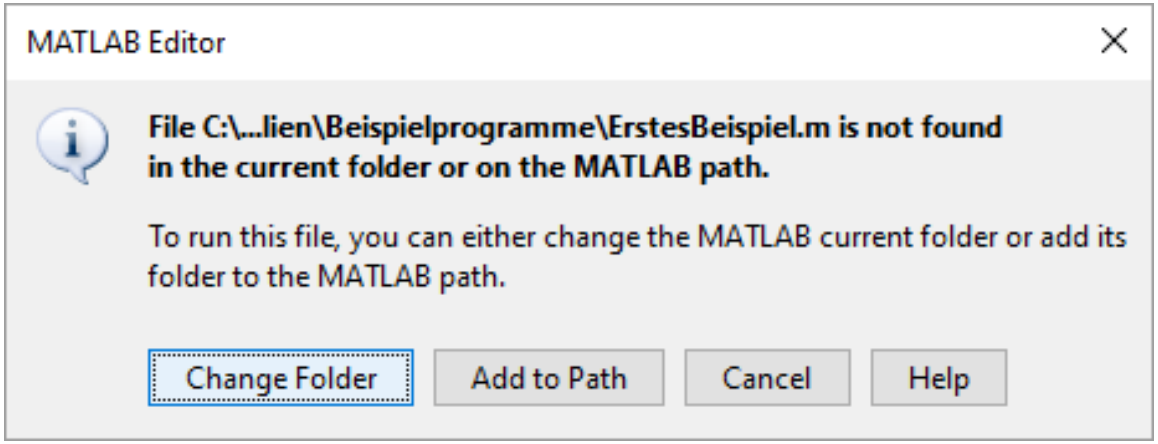
- Beispiel: MATLAB hat einen Befehl `sqrt` (Wurzelfunktion). Wenn jemand seine Datei `sqrt.m` nennt, steht der originale MATLAB-Befehl `sqrt` nicht mehr zur Verfügung!
- Kein Grund zur Panik: Es genügt, die Datei umzubenennen, z.B. in `my_sqrt.m`. Damit ist das Problem gelöst.

Skripte ausführen

Ein abgespeichertes Skript kann mittels „Run“-Knopf (im Register *EDITOR*) ausgeführt werden:

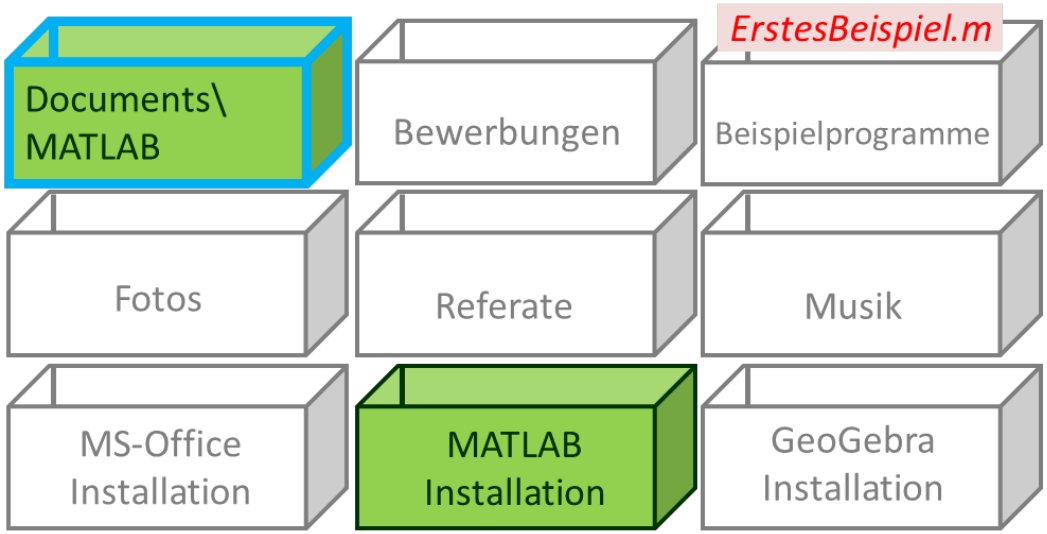


Hups?
Beim ersten Mal kann (sollte) so etwas passieren:



Skripte ausführen

Beispiel einer Verzeichnisstruktur:

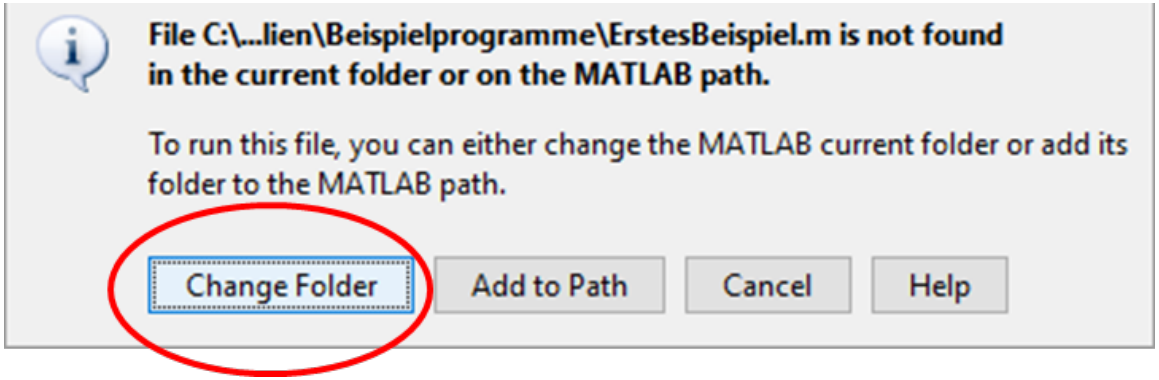


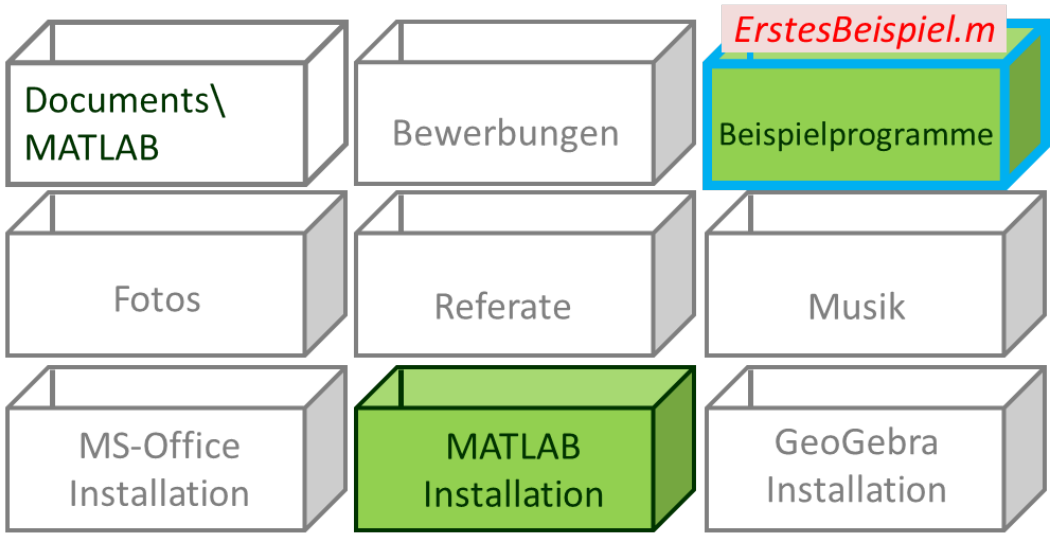
- Grün: Verzeichnisse, in denen MATLAB nach ausführbaren Dateien sucht (enthalten im *Suchpfad* oder *Path* auf Englisch).
- Blauer Rahmen: das aktuelle Arbeitsverzeichnis. Beim Start in Windows-Systemen ist das defaultmäßig `Benutzer\Documents\MATLAB`.
- Grau: Für MATLAB „unsichtbare“ Verzeichnisse.

Das heißt: Falls die Datei `ErstesBeispiel.m` nicht in einem Verzeichnis des Suchpfades (in einem der grünen Verzeichnisse) liegt, kann MATLAB diese nicht ausführen!

Skripte ausführen

1. Möglichkeit (empfohlen): Das Arbeitsverzeichnis wechseln (*Change Folder*).

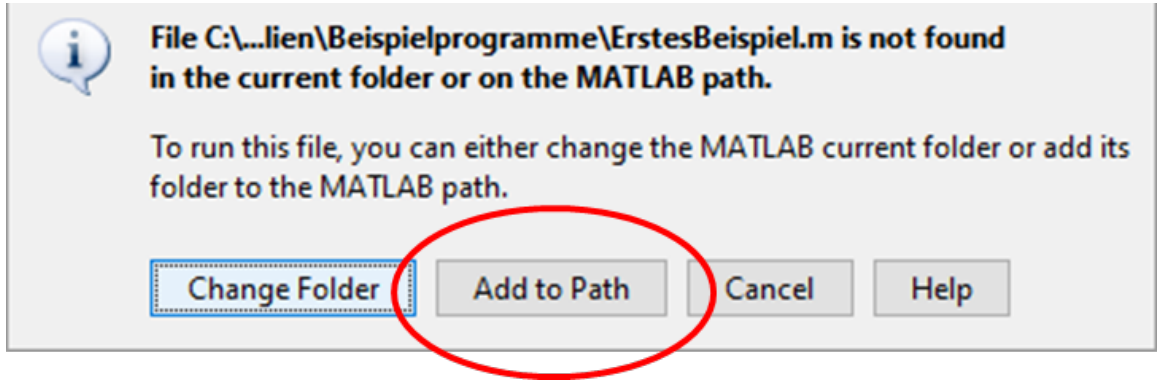


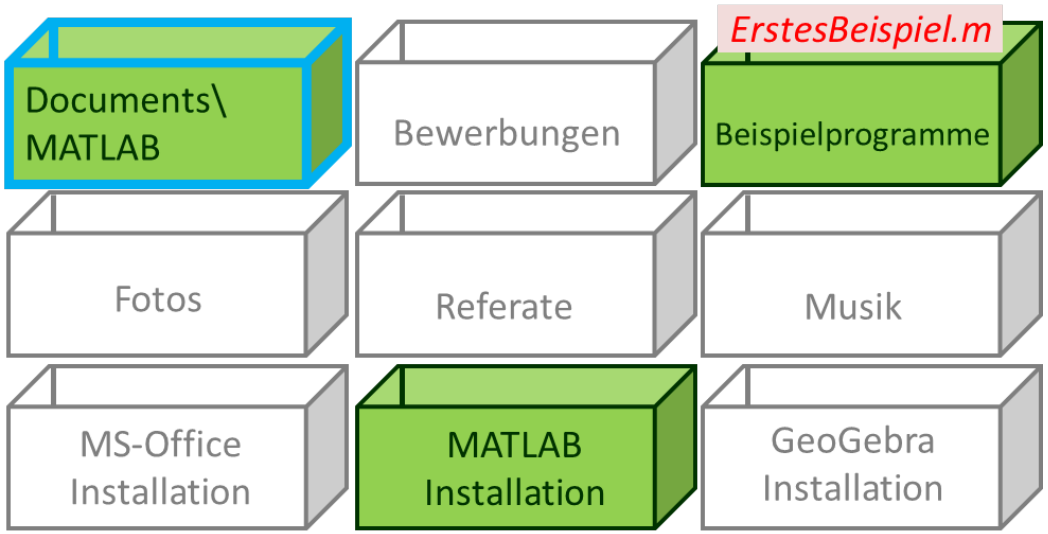


- Das Programm ErstesBeispiel.m ist jetzt sichtbar und kann ausgeführt werden.
- Das Verzeichnis Beispielprogramme ist jetzt das Arbeitsverzeichnis.
- Die Programme in dem früheren Arbeitsverzeichnis (Documents\MATLAB) sind unsichtbar.

Skripte ausführen

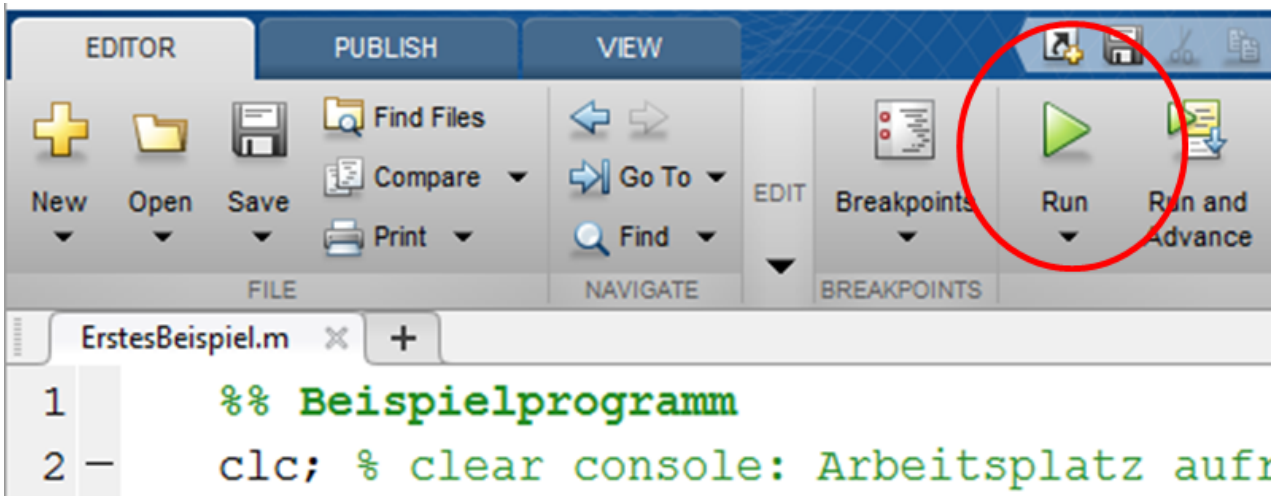
2. Möglichkeit:
Das Verzeichnis zum Pfad hinzufügen (*Add To Path*).





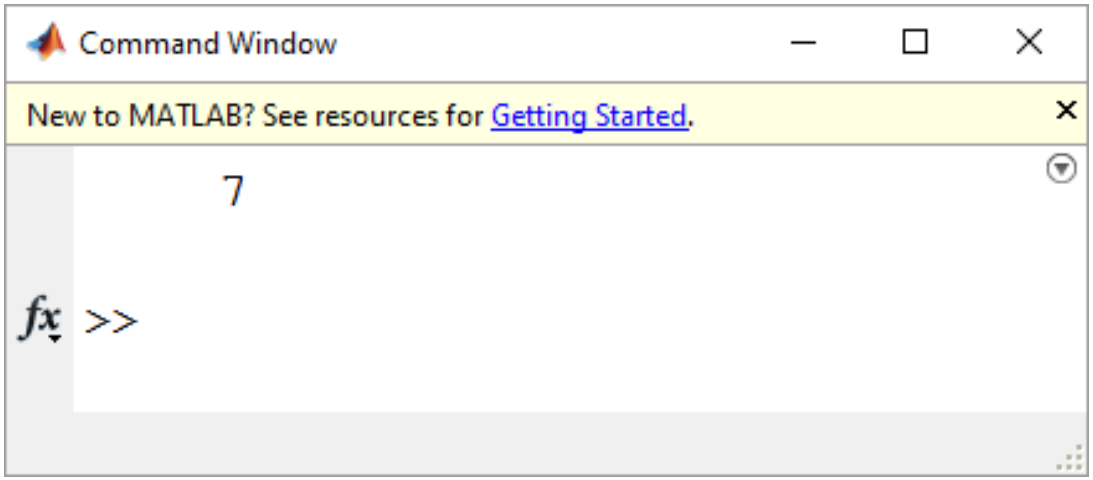
- Das Programm ErstesBeispiel.m ist jetzt sichtbar und kann ausgeführt werden.
- Das Arbeitsverzeichnis bleibt Documents\MATLAB.
- Die Programme in den beiden Verzeichnissen (Documents\MATLAB und Beispielprogramme) sind sichtbar.

Skripte ausführen



Im Register *EDITOR* den „Run“-Knopf anklicken!

Das Ergebnis wird im *Command Window* angezeigt.



Hat alles auf Anhieb funktioniert? Herzlichen Glückwünsch!

Meistens (sogar normalerweise) jedoch kommen irgendwelche (mehr oder weniger verständliche) Fehlermeldungen.

Ein Programmierer verbringt nicht die meiste Zeit mit Tippen von Quelltexten (es gibt sogar Programmierer, die nur vergleichweise langsam tippen können. . .) sondern mit der Fehlersuche. Dieser Prozess wird auch *Debugging* genannt.

Unvermeidlich: Fehlersuche – Debuggen

Hier einige der häufigsten (nicht nur Anfänger-)Fehler (diese erscheinen dann in rot anstatt der erwarteten „7“ im *Command Window*):

- Undefined function or variable 'Y'.
  Error in ErstesBeispiel (line 8)
  z = x + Y;

  Eine Variable (hier: Y) wurde nicht gefunden. Häufigster Grund: Tippfehler in Namen (bei einem Buchstaben eher unwahrscheinlich) oder Nichtbeachten der Groß-Kleinschreibung. Y und y sind unterschiedliche Variablen!
- Undefined function or variable 'dispp'.
  Error in ErstesBeispiel (line 11)
  dispp(z);

  Die gleiche Fehlermeldung erscheint auch, wenn ein MATLAB-Befehl fehlerhaft geschrieben worden ist, weil MATLAB nicht weiß, was gemeint war, also undefiniert ist (englisch undefined).

Fehlersuche – Debuggen

- Das hier ist etwas tückischer:

  ```
  >> Erstes Beispiel
  Undefined function or variable 'Erstes'.
  ```

  Der Dateiname ist ungültig! Das Programm sollte unter einem gültigen Namen (hier: ohne Leerzeichen) gespeichert werden.

- ```
  Error: File: ErstesBeispiel.m Line: 8 Column: 7
  The expression to the left of the equals sign is not a valid
  target for an assignment.
  ```

  Schauen wir die Zeile 8 an. Hier steht:

  ```
  x + y = z;
  ```

  Das ist nicht erlaubt. In der Mathematik ist es egal, ob wir $z = x + y$ oder $x + y = z$ schreiben. Beim Programmieren hat das Zeichen „=“ eine andere Bedeutung. Es ist eine *Zuweisung*: Die Variable *links* davon erhält den Wert, welcher *rechts* berechnet worden ist.
  ```

Fehlersuche – Debuggen

- Und wenn zuviel angezeigt wird?

```
x =
     2
y =
     5
z =
     7
     7
```

Das hier ist keine Fehlermeldung. Unnötige Anzeigen können ein Programm jedoch erheblich verlangsamen. Wenn am Ende einer Zeile das Zeichen „;“ fehlt, wird das Ergebnis der Zeile angezeigt.

1. Die ersten Schritte

Karlsruher Institut für Technologie

Mein erstes MATLAB Programm

Mit Buchstaben rechnen: Variablen

- Was ist eine Variable?
- Variablentypen
  - Zahlen und Zeichenketten
  - Zahlen und Zeichenketten kombinieren
- Anwendungsbeispiel: Benutzereingabe

Was ist eine Variable?

Das Programm `ErstesBeispiel.m` verwendet Buchstaben `x`, `y` und `z`. In der Programmierung nennt man solche Buchstaben, die verschiedene Werte annehmen können, *Variablen*.

- Eine *Variable* ist ein konkreter Speicherbereich (auf einem Rechner), der Werte aufnehmen kann und unter dem Namen angesprochen wird. Sie ist KEINE Unbestimmte oder Lösungsvariable einer Gleichung.
- Keine Deklaration (Unterschied zu z.B. C++)
- Variablennamen:
  - Buchstabe, gefolgt von beliebig vielen Buchstaben oder Ziffern oder einem Unterstrich „_“ (kein „-“, „+“)
  - Unterscheidung zwischen Groß- und Kleinschreibung
- `x = 3*x+2` ist keine Gleichung, sondern eine *Wertzuweisung*. Eventuell vorhandene Inhalte im Speicherbereich `x` werden ohne Warnung überschrieben.

Was ist eine Variable?

Hier ein kleines Beispielprogramm (Variablen.m) mit zwei Variablen hilfsvariable und y. Führt es aus, um zu sehen, was passiert:

```
%% Beispielprogramm zum Thema Variablen
clc; clear all; close all;
%% Variablen festlegen
hilfsvariable = 3;
y = hilfsvariable; % y = 3
y = y.^2; % y = 3^2 = 9
disp(y);
```

Variablentypen

- Die einfachsten Variablentypen sind:
  - Zahlen (double)
  - Zeichenketten (string)
  - Vektoren (array)
- In den meisten „klassischen“ Programmiersprachen wie C, C++, Java, . . . müssen die Variablentypen explizit deklariert werden. In MATLAB entfällt dieser Schritt.
- MATLAB erkennt den Datentyp automatisch.
  - Alle Zahlen sind in MATLAB erstmal *double* (15-16 signifikante Stellen).
  - Um die Effektivität der Programme zu verbessern, können auch andere Datentypen verwendet werden. Das muss dann aber gut überlegt werden!

Zahlen und Zeichenketten kombinieren

Karlsruher Institut für Technologie

Bisher waren unsere Programmausgaben wenig informativ. Es ist jedoch wichtig, dass der Benutzer erkennt, was gerade angezeigt wird. Führt das folgende Programm VerstaendlicheAusgabe.m aus:

```
clc; clear all; close all;
Figur = 'Kreis'; % Figurenname ist vom Typ string
Radius = 2; % Radius ist vom Typ double
Flaeche = pi*(Radius.^2); % MATLAB kennt "pi"
% Anzeige:
% Zwei Strings werden "zusammengeklebt":
disp([Figur,'flaeche berechnen:']);
% String kann nicht mit double zusammengeklebt werden;
% deshalb wandeln wir die Zahl "Radius" mittels
% Befehl num2str in einen String um:
disp(['Radius = ',num2str(Radius)]);
disp(['Flaeche = ',num2str(Flaeche)]);
```

Der Befehl **num2str** (*number to string*) erlaubt „Äpfel“ (Strings) mit „Birnen“ (Doubles) zusammenzukleben.

Anwendungsbeispiel: Benutzereingabe

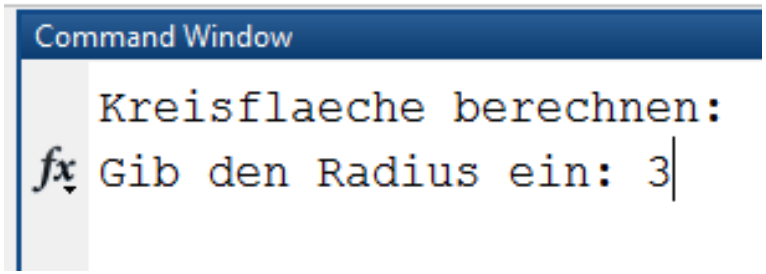
Der Befehl **input** ermöglicht es dem Benutzer, den Radius des Kreises selbst einzugeben (Programm BenutzerEingabe.m):

```
clc; clear all; close all;
Figur = 'Kreis';
% Zuerst Benutzer informieren, was gemacht wird:
disp([Figur,'flaeche berechnen:']);
% Benutzer auffordern, den Radius einzugeben:
Radius = input('Gib den Radius ein: ');
Flaeche = pi*(Radius.^2);
% Ergebnis anzeigen:
disp(['Flaeche = ',num2str(Flaeche)]);
```

Anwendungsbeispiel: Benutzereingabe

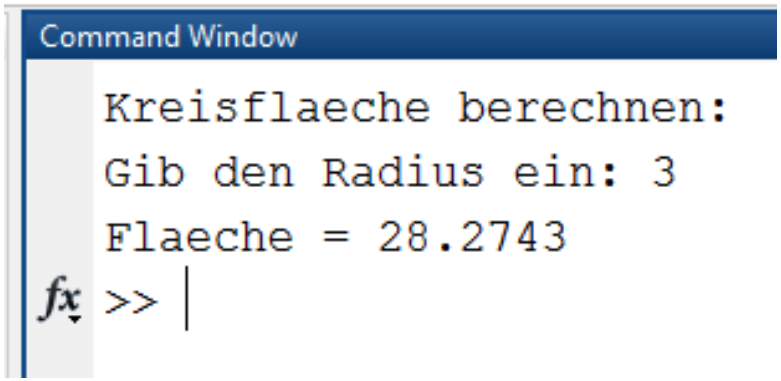
Startet das vorherige Programm! Zuerst passiert. . . nichts. . .
Das Programm wartet auf die Benutzereingabe!

- Wechselt zum *Command Window*.
- Parameterwerte werden über die Tastatur im *Command Window* eingegeben:



Bestätigt die Eingabe mit *Enter*/*Eingabe*-Taste auf der Tastatur.

- Danach wird das Programm weiter ausgeführt:



Karlsruher Institut für Technologie

MINT-KOLLEG BADEN-WÜRTTEMBERG

Einfach Programmieren lernen mit MATLAB!

2. Etwas Mathematik

MINT-Kolleg Baden-Württemberg

2. Etwas Mathematik

Karlsruher Institut für Technologie

Arithmetische Operatoren und mathematische Funktionen
- Arithmetische Operatoren
- Mathematische Funktionen

Viele Zahlen auf einmal: Vektoren in MATLAB

Visualisierung geometrischer Figuren

Arithmetische Operatoren

+	Addition
-	Subtraktion
.*	Multiplikation
./	Division
.\	Division (a\b=b/a)
.^	Potenzierung
( )	Gruppierung

Empfehlung: Operatoren

.* ./ .\ .^

mit Punkt benutzen AUSSER man will explizit MATRIX-Algebra verwenden!

Beispiel

Folgende Befehle könnt ihr direkt im *Command-Window* ausprobieren:

```
>> 1+2.*3
ans = 7
>> 1-2\6./3    % Dies enstpricht 1-(6/2)/3
ans = 0
>> 2.^3.^2    % Entspricht (2.^3).^2 = 2.^(3.*2)
ans = 64
>> 2.^(3.^2)    % 2^9
ans = 512
```

Mathematische Funktionen

- Trigonometrische Funktionen:
  - sin, cos, tan, cot (Argument im Bogenmass)
  - sind, cosd, tand, cotd (Argument im Gradmass)
- Wurzelfunktion: sqrt
- Rundung zur nächstgelegenen ganzen Zahl: round

Beispiel
Folgende Befehle könnt ihr direkt im *Command-Window* ausprobieren:

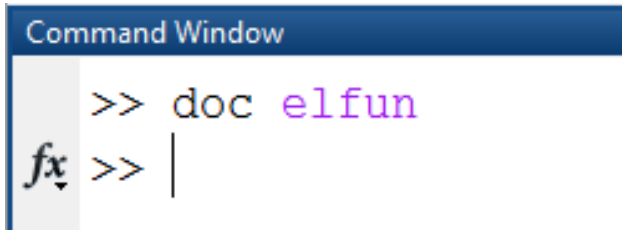
```
>> sin(pi/6)
ans = 0.5000
>> cosd(90)
ans = 0
```

```
>> sqrt(81)
ans = 9
>> round(-4.6)
ans = -5
```

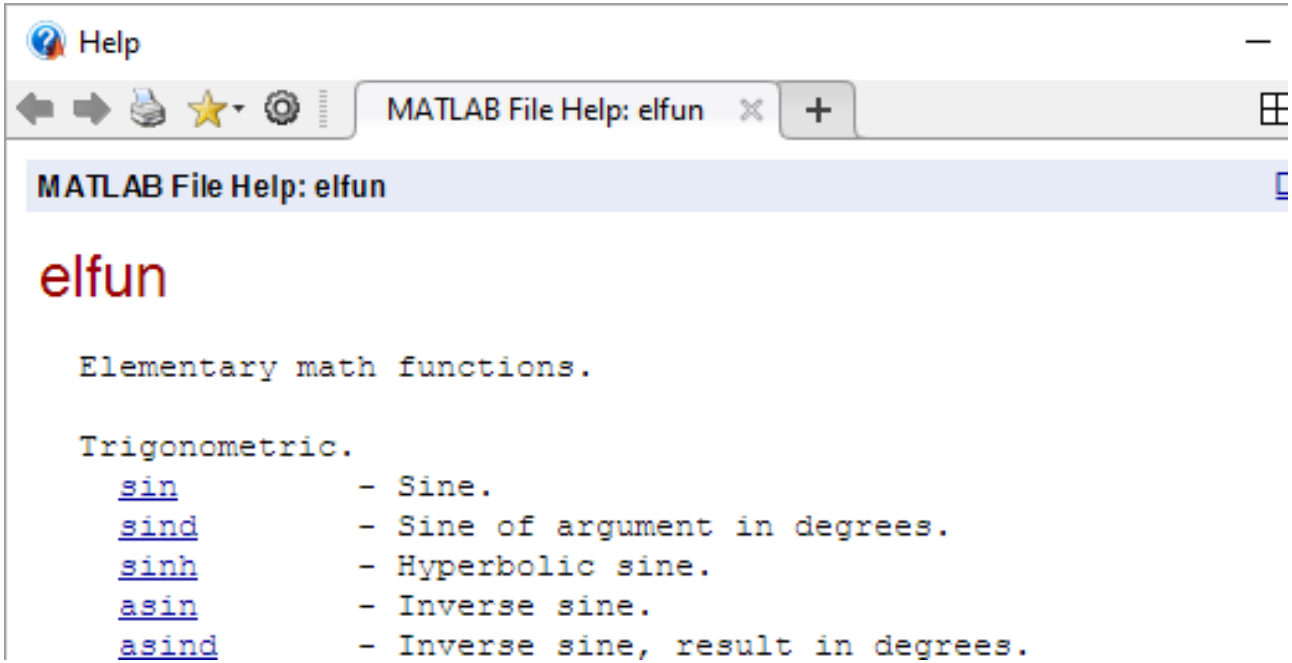
Mathematische Funktionen

Übersicht aller elementaren Funktionen:

- doc elfun
  im *Command-Window* eingeben:



- Ein *Help*-Fenster mit einer Auflistung wird geöffnet:



2. Etwas Mathematik

Arithmetische Operatoren und mathematische Funktionen

Viele Zahlen auf einmal: Vektoren in MATLAB

- Vektoren mit wenigen Einträgen
- Mathematische Operatoren und Funktionen
- Automatisch erzeugte Vektoren

Visualisierung geometrischer Figuren

Vektoren mit wenigen Einträgen

Karlsruher Institut für Technologie

Oft ist es praktisch, viele Zahlen in einer Variablen zusammenzufassen. Solche Variablen nennt man *arrays* oder *Vektoren* (in MATLAB).

- Dafür verwendet man die eckigen Klammern [ ]. Die einzelnen Einträge werden mit Kommata getrennt, wie im Beispiel unten gezeigt wird (Programm Vektoren.m):

```
clc; clear all; close all;
x = [12, 13, 11]; % x ist ein (Zeilen-)Vektor
disp(x);
```

- Einzelne Einträge werden mit Hilfe runder Klammern erreicht:

```
disp(x(2)); % Zeige den 2. Eintrag x(2) an: d.h., 13
a = x(3)-x(1)./4; % a = 11-12/4
disp(a)
```

Mathematische Operatoren und Funktionen

Weiter im Programm Vektoren.m:

- Mathematische Operatoren und Funktionen greifen gleich auf alle Einträge des Vektors zu:

```
y = x+3; % y(1)=x(1)+3, y(2)=x(2)+3, y(3)=x(3)+3
disp(y); % d.h. y = [12+3, 13+3, 11+3]
z = x.^2; % z(1)=x(1).^2, z(2)=x(2).^2, z(3)=x(3).^2
disp(z); % d.h., z = [12^2, 13^2, 11^2]
```

- Zwei (und mehrere) Vektoren können auch zusammengesetzt werden:

```
u = [x, z]; % Vektoren x und z werden zusammengeklebt
disp(u); % d.h. u = [12, 13, 11, 144, 168, 121]
```

Automatisch erzeugte Vektoren

Sehr oft (meistens) werden Vektoren automatisch erzeugt.
Einen Vektor mit gleichen Abständen zwischen den Einträgen erzeugt man mit dem Doppelpunkt : nach der Syntax

x = Startwert:Schrittweite:Endwert

Die Schrittweite wird auch *Inkrement* genannt.

Beispiel (Programm AutoVektoren.m):

- Programmanfang wie immer:

```
clc; clear all; close all;
```

- Erzeugt einen Vektor mit x(1)=2 und weiter mit Schrittweite 1, bis der Wert 8 erreicht wird:

```
x = 2:1:8;
disp(x); % x=[2, 3, 4, 5, 6, 7, 8]
```

Automatisch erzeugte Vektoren

Weiter im Programm AutoVektoren.m:

- Schrittweite kann unterschiedlich, auch rückwärts (dann negativ setzen) gewählt werden:

```
y = -3:2:5; % Schrittweite 2
disp(y); % y=[-3, -1, 1, 3, 5];
z = 4:-0.5:1; % Schrittweite 0.5, RUECKWAERTS!
disp(z); % z=[4, 3.5, 3, 2.5, 2, 1.5, 1]
```

- Der angegebene Endwert wird nie überschritten:

```
u = 1:3:8;
disp(u); % u=[1, 4, 7];
v = 8:-3:1;
disp(v); % v=[8, 5, 2];
```

Bemerkung: Falls der Startwert oder Endwert wichtiger als die Schrittweite sind (hängt von der Anwendung ab!), wird besser der Befehl linspace verwendet. Gebt doc linspace im *Command-Window* ein, um mehr zu erfahren.

2. Etwas Mathematik

Arithmetische Operatoren und mathematische Funktionen

Viele Zahlen auf einmal: Vektoren in MATLAB

Visualisierung geometrischer Figuren

Visualisierung geometrischer Figuren

- Um Figuren in der Ebene zu zeichen, kann der Befehl fill verwendet werden. Die MATLAB-Syntax lautet:

```
fill(XKoord,YKoord,FuellFarbe,'EdgeColor',RandFarbe)
```

- XKoord: Vektor mit den *x*-Koordinaten der Eckpunkte
- YKoord: Vektor mit den *y*-Koordinaten der Eckpunkte
- Für FuellFarbe und RandFarbe sind folgende Werte möglich: 'yellow', 'magenta', 'cyan', 'red', 'green', 'blue', 'white', 'black'
- Die Koordinaten (Achsen) in dem Figure-Fenster werden mit Hilfe des Befehls axis festgelegt. Die MATLAB-Syntax lautet:

```
axis([xLinks, xRechts, yLinks, yRechts])
```

- Falls man mehrere Figuren im gleichen Bild anzeigen möchte, sollte man den Befehl hold on verwenden. Dann werden vorhandene grafische Objekte nicht gelöscht, wenn ein neues Objekt erzeugt wird.

Beispiel: Programm GeomFiguren.m

- Vorbereitung des Grafikfensters:

```
clc; clear all; close all;
figure(1); % oeffne ein Grafikfenster
axis equal
axis([-5, 5, -10, 10]) % x-Achse: -5 -> 5, y-A.: -10 -> 10
grid on % Gitterlinien anzeigen
hold on % lasse alle Grafiken uebereinander
```

- Malt ein Viereck mit Eckpunkten $P_1(-3;2)$, $P_2(-1;2)$, $P_3(-1;4)$, $P_4(-3,4)$, gelber Füllung und rotem Rand:

```
xP = [-3, -1, -1, -3]; % x-Koordinaten der Eckpunkte
yP = [2, 2, 4, 4]; % y-Koordinaten der Eckpunkte
fill(xP,yP,'yellow','EdgeColor','red')
```

- Malt ein Dreieck mit Eckpunkten $Q_1(-3;-8)$, $Q_2(3;-8)$, $Q_3(0;-4)$, türkiser Füllung und blauem Rand

```
xQ = [-3, 3, 0]; % x-Koordinaten der Eckpunkte
yQ = [-8, -8, -4]; % y-Koordinaten der Eckpunkte
fill(xQ,yQ,'cyan','EdgeColor','blue')
```

Beispiel: Programm GeomFiguren.m (Fortsetzung)

- Ein Kreis wird als ein „Vieleck“ mit sehr, sehr vielen Ecken angezeigt. Der Rand eines Kreises mit dem Mittelpunkt im Punkt $(x_c; y_c)$ und Radius R kann wie folgt *parametrisiert* werden:

$$x(t) = x_c + R\cos(t) \quad , \quad y(t) = y_c + R\sin(t) \quad , \quad t \in [0; 2\pi[$$

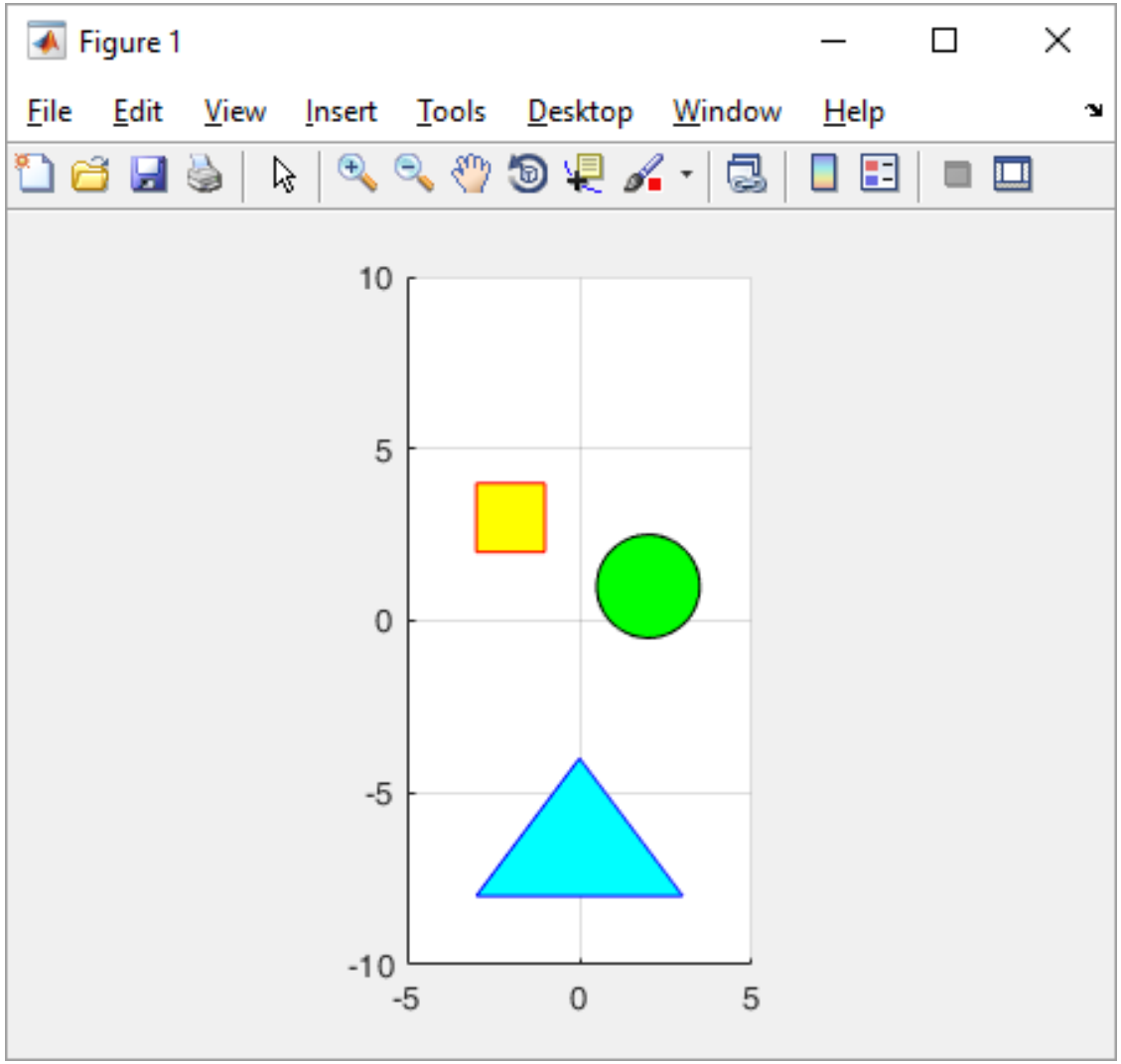
  Vorgehensweise:

  - Generiere zuerst einen Vektor mit den Werten des *Parameters t*.
  - Berechne die entsprechenden Werte von *x* und *y* (zwei Vektoren!).
  - Verwende die berechneten *x*- und *y*-Werte als Ecken eines Polygons.

```
xc = 2; yc = 1; % Mittelpunkt
R = 1.5;
% Randpunkte generieren:
t = 0:0.1:2*pi;
xk = xc+R.*cos(t);
yk = yc+R.*sin(t);
fill(xk,yk,'green','EdgeColor','black')
```

Beispiel: Programm GeomFiguren.m (Fortsetzung)

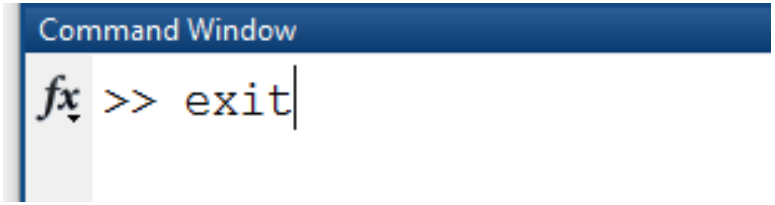
Das Ergebnis:



Zusammenfassung

- MATLAB ist eine von vielen Programmiersprachen.
- Skripte sind Aneinanderreihungen von MATLAB-Befehlen. Es gibt gewisse Regeln, die man bei der Namensgebung beachten sollte.
- Ihr kennt jetzt den *Editor* und das *Command Window*.
- Programme verwenden *Variablen*. Ihr habt bisher drei Typen kennengelernt: Zahlen (double), Zeichenketten (string) und Vektoren (array).
- Ein Benutzer kann die Werte der Variablen auch eingeben.
- Mathematische Operatoren und Funktionen wirken auf ganze Vektoren.
- Mit dem Befehl `fill` können geometrische Figuren gezeichnet werden.

Eine Pause ist jetzt verdient. Gebt `exit` im *Command-Window* ein:



Karlsruher Institut für Technologie

MINT-KOLLEG BADEN-WÜRTTEMBERG

Einfach Programmieren lernen mit MATLAB!

3. Ablaufstrukturen I

MINT-Kolleg Baden-Württemberg

3. Ablaufstrukturen I

„Richtiges“ Programmieren: Ablaufstrukturen

Idee der wichtigsten Ablaufstrukturen

Verzweigungen

Programmstrukturen

In fast allen Programmiersprachen:

- Die *Kommunikation* mit der Außenwelt:
  - Ein- und Ausgabe von Zahlen und Text (Befehle `input` und `disp`)
  - Grafikdarstellung (Befehl `fill`)
  - Dateioperationen
- Die zentralen *Ablaufstrukturen*:
  - Verzweigung (Auswahl, Alternative)
  - Schleife (Iteration, Wiederholung)
  - Funktion (Aufruf weiterer Funktionen innerhalb eines Programms oder einer Funktion)
- Die erweiterten *Datentypen*:
  - Feld
  - (Struktur)
  - String

Idee der wichtigsten Ablaufstrukturen

Ablaufstrukturen werden benötigt, um den Programmablauf zu steuern.

Die *Sequenz* bildet die normale Reihenfolge des Ablaufs von Anweisungen in einem Computerprogramm.

Hier ein kleines Beispiel in Pseudocode aus dem Alltag:

```
Gehe bis zur Straße
Halte an
Schaue nach links und rechts
Gehe über die Straße
```

Wenn man die Situation genauer betrachtet, ist dieses Programm noch nicht vollständig: Nur nach links und rechts zu schauen, reicht nicht, um sicher die Straße zu überqueren – man muss auch sicherstellen, dass die Straße wirklich frei ist!

Ablaufstrukturen: Verzweigungspunkte (Auswahl, Alternative)

Mit der folgenden Verbesserung unseres Programms kann man schon das Leben des Passanten retten:

```
Gehe bis zur Straße
Halte an
Schaue nach links und rechts
Falls die Straße frei ist {
   Gehe über die Straße
}
```

Die rot markierte Zeile beschreibt eine *Verzweigung*.
Jetzt ist unser Passant schon in Sicherheit. Es kann jedoch passieren, dass er/sie die Straße nicht überquert, sondern für immer dort stehen bleibt!
Falls die Straße nicht frei ist, wäre es sinnvoll, nach einem kurzen Moment nochmals zu schauen, ob die Straße frei ist!

Ablaufstrukturen: Schleifen (Iteration, Wiederholung)

Jetzt sollte es endlich funktioneren:

```
Gehe bis zur Straße
Halte an
Setze die Marke gehe=nein
Wiederhole solange gehe=nein {
   Schaue nach links und rechts
   Falls die Straße frei ist {
      Setze die Marke gehe=ja
      Gehe über die Straße
   }
}
```

Die rot markierten Zeilen bilden eine *Schleife* mit einem *Abbruchkriterium*.
Straßen werden sehr oft überquert. Irgendwann kann es ein Mensch (mehr oder weniger) „automatisch“.

Ablaufstrukturen: Funktionen

Hauptprogramm

Gehe bis zur Straße
Führe die Funktion ueberquere_eine_straße aus

Funktion ueberquere_eine_straße

```
ueberquere_eine_straße {
   Halte an
   Setze die Marke gehe=nein
   Wiederhole solange gehe=nein {
      Schaue nach links und rechts
      Falls die Straße frei ist {
         Setze die Marke gehe=ja
         Gehe über die Straße
      }
   }
}
```

Da der Schritt „überquere eine Straße“ sehr oft benötigt wird, lohnt es sich, ihn in eine handliche, wiederverwendbare *Funktion* auszulagern.

3. Ablaufstrukturen I

„Richtiges“ Programmieren: Ablaufstrukturen

Verzweigungen

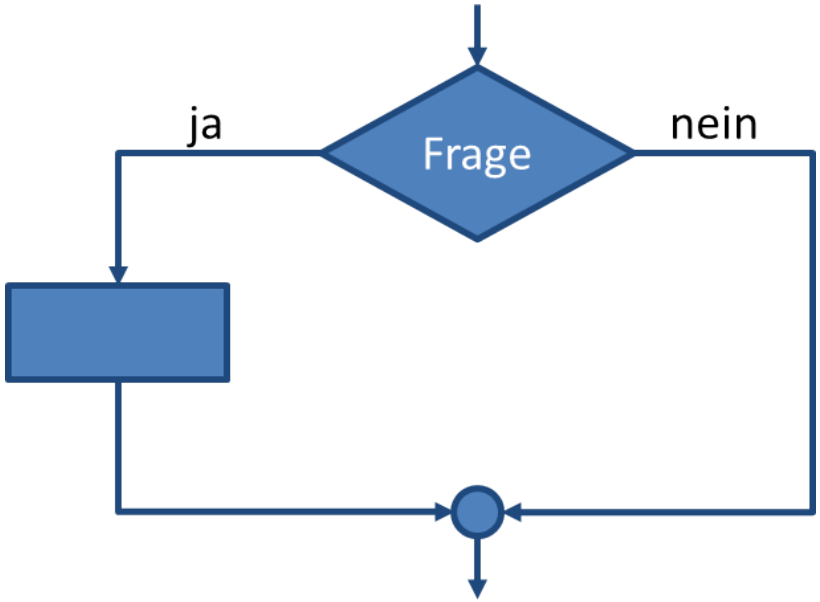
- Einseitige und zweiseitige Auswahl
- Logische Ausdrücke
- Beispiel: Programm FlaechenRechner.m

Verzweigungen: einseitige Auswahl

"Wenn die Bedingung erfüllt ist, soll Folgendes gemacht werden."

MATLAB-Syntax:

```
if logischerAusdruck
    Anweisungen
end
```



Probiert in einem Skript die folgenden Beispiele aus!

Beispiel:

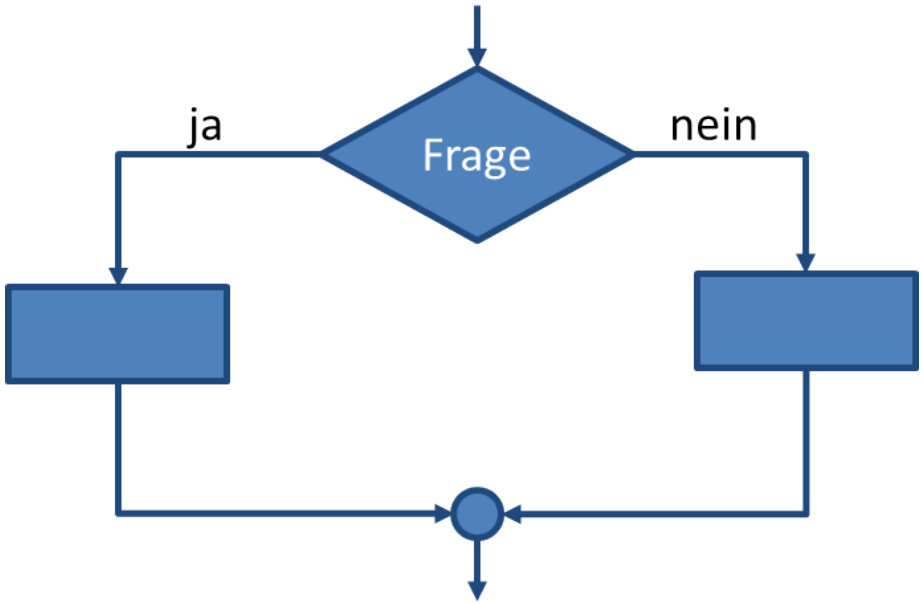
```
n = input('Gib eine Zahl ein: ');
if n<5
    disp('Die Zahl ist kleiner als 5.')
end
```

Verzweigungen: zweiseitige Auswahl

Bei Erfüllung bzw. Nichterfüllung einer Bedingung erfolgen unterschiedliche Aktionen.

MATLAB-Syntax:

```
if logischerAusdruck
    Anweisungen
else
    Anweisungen
end
```



Beispiel:

```
n = input('Gib eine Zahl ein: ');
if n<5
    disp('Die Zahl ist kleiner als 5.')
else
    disp('Die Zahl ist gleich oder groesser als 5.');
end
```

Logische Ausdrücke

Die *Frage* in Verzweigungen muss ein *logischer Ausdruck* sein, d.h., eine Behauptung, die entweder *erfüllt (wahr)* oder *nicht erfüllt (falsch)* ist.

Hier die wichtigsten Beispiele:

- *gleich:* == (zwei Gleichheitszeichen „=“) und
  ungleich: ~= („Tilde“ und Gleichheitszeichen „=“)

```
5==4; % falsch
```

```
3~=6; % wahr
```

- *größer:* >, *größer oder gleich:* >=, *kleiner:* <, *kleiner oder gleich:* <=

```
4>=7; % falsch
```

```
5<9; % wahr
```

Bemerkung: Die Ausdrücke können auch miteinander kombiniert werden. Schaut die Hilfe zu den Befehlen `and`, `or` und `xor` dafür an. Hier brauchen wir diese Befehle jedoch noch nicht!

Beispiel: Programm FlaechenRechner.m

- Dieses Programm berechnet die Fläche eines Kreises, eines Rechteckes oder eines rechtwinkliges Dreiecks, je nach dem, was der Benutzer ausgewählt hat.
- Die Auswahl erfolgt mit Hilfe von Zahlen: 1 bedeutet Kreis, 2: Rechteck, 3: rechtwinkliges Dreieck.
- Falls eine dieser Zahlen eingegeben wird, wird der Benutzer nach weiteren Parametern gefragt (beim Kreis: Radius; Rechteck: beide Seitenlängen; rechtwinkliges Dreieck: beide Katheten). Danach wird die Fläche der entsprechenden Figur ausgegeben.
- Falls die Auswahl weder 1, noch 2, noch 3 ist, d.h., die Benutzereingabe ungültig ist, wird eine entsprechende Warnung angezeigt.
- Zur Erläuterung des Aufbau des Programms:
  - Zuerst wird nur die Struktur des Programms gezeigt.
  - Danach werden die Details einzelner Blöcke beschrieben (in der Struktur mit Großbuchstaben markiert).

Versucht, das Programm nachzuvollziehen und zum Laufen zu bringen!

Struktur des Programms

```
clc; clear all; close all;
disp('Programm berechnet die Flaecheninhalte:')
disp('  1:Kreis, 2:Rechteck, 3:Rechtwinkliges Dreieck');
auswahl = input('Waehle eine Figur (1-3) aus: ');

eingabe_gueltig = 0; % gueltige Eingabe: 0=falsch, 1=wahr
if auswahl==1 % Eingabe gueltig, berechne Kreis
    eingabe_gueltig = 1;
    EINGABE UND BERECHNUNG KREIS
end
if auswahl==2 % Eingabe gueltig, berechne Rechteck
    eingabe_gueltig = 1;
    EINGABE UND BERECHNUNG RECHTECK
end
if auswahl==3 % Eingabe gueltig, berechne Dreieck
    eingabe_gueltig = 1;
    EINGABE UND BERECHNUNG DREIECK
end
ERGEBNIS ANZEIGEN
```

Blöcke

EINGABE UND BERECHNUNG KREIS

```
Figur = 'Kreis';
Radius = input('Gib den Radius ein: ');
Flaeche = pi.*(Radius.^2);
```

EINGABE UND BERECHNUNG RECHTECK

```
Figur = 'Rechteck';
a = input('Gib die Laenge der ersten Seite ein: ');
b = input('Gib die Laenge der zweiten Seite ein: ');
Flaeche = a.*b;
```

EINGABE UND BERECHNUNG DREIECK

```
Figur = 'Rechtwinkliges Dreieck';
a = input('Gib die Laenge der ersten Kathete ein: ');
b = input('Gib die Laenge der zweiten Kathete ein: ');
Flaeche = (a.*b)./2;
```

Blöcke (Fortsetzung)

ERGEBNIS ANZEIGEN

```
%% Ergebnis anzeigen
if eingabe_gueltig==0
    % Falls keine der Bedingungen oben erfuellt wurde,
    % ist eingabe_gueltig gleich 0 geblieben.
    disp('Eingabe ungueltig! Es wurde nichts berechnet!')
else
    disp(['Figur: ',Figur]);
    disp(['Flaeche = ',num2str(Flaeche)]);
end
```

Funktioniert alles? Dann versucht, eine weitere Figur hinzufügen (zum Beispiel ein Trapez)!

Karlsruher Institut für Technologie

MINT-KOLLEG BADEN-WÜRTTEMBERG

Einfach Programmieren lernen mit MATLAB!

4. Ablaufstrukturen II

MINT-Kolleg Baden-Württemberg

4. Ablaufstrukturen II

Karlsruher Institut für Technologie

Schleifen (while-Schleife)

Schleifen und Verzweigungen kombinieren

Schleifen (while-Schleife)

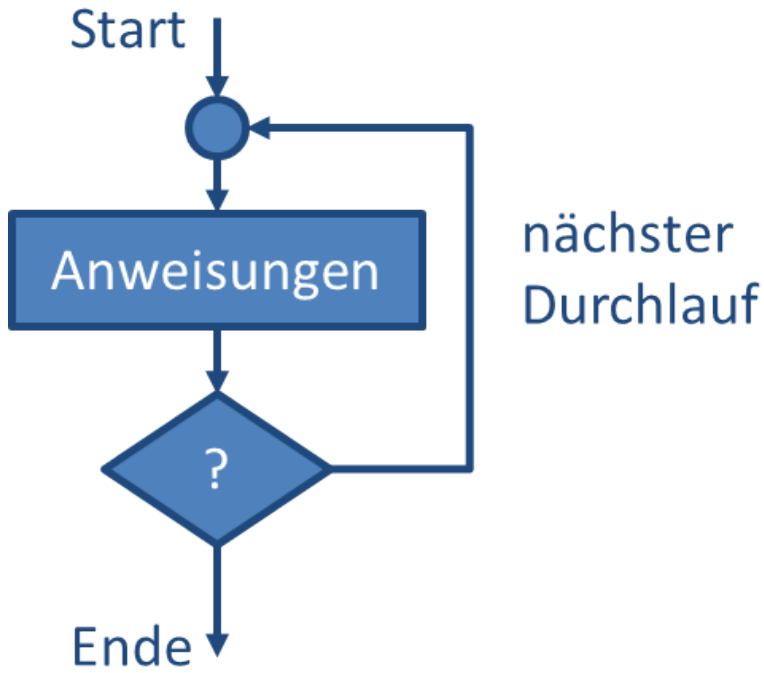
Schleifen werden benötigt, um Vorgänge zu automatisieren. In MATLAB gibt es zwei Schleifenarten: while-Schleifen (Wiederholschleifen) und for-Schleifen (Zählschleifen).

MATLAB-Syntax (while-Schleife):

```
while logischerAusdruck
    Anweisungen
end
```

Beispiel:

```
a=5; b=30;
while (a<b)
    b=b/2;
    disp(b);
end
```



Beispiel einer typischen Anwendung: Wiederholungen, bis der Benutzer den Prozess beenden möchte (To continue? y/n).

Beispiel

Folgendes Beispielprogramm Schleifen.m berechnet die Fläche und den Umfang immer größer werdender Kreise. Probiert es aus!

```
clc; clear all; close all;
R_min = 0.5; % der kleinste Radius
R_max = 2; % der groesste Radius
h = 0.3; % Schrittweite, mit der der Radius vergroessert wird

disp('Kreisflaeche und Umfang');
disp('R: Radius, F: Flaeche, U: Umfang');

R = R_min; % Initialisierung
while R <= R_max
  F = pi.*R.^2; % Flaeche berechnen
  U = 2.*pi.*R; % Umfang berechnen
  % Anzeige:
  disp(['R=',num2str(R),' F=',num2str(F),' U=',num2str(U)]);
  R = R + h; % R wird um die Schrittweite h vergroessert
end
```

Abbrechen mit Strg+C

Insbesondere im Zusammenhang mit Schleifen ist die Tastenkombination

Strg+C

hilfreich, mit der man die Ausführung von MATLAB-Programmen selbst abbrechen kann.

4. Ablaufstrukturen II

Karlsruher Institut für Technologie

Schleifen (while-Schleife)

Schleifen und Verzweigungen kombinieren

Schleifen und Verzweigungen kombinieren

Schleifen und Verzweigungen können miteinander (fast) beliebig kombiniert werden.

Beispiel: Ein Würfelspiel-Simulator
(Programm `Wuerfel.m`)

- Benutzer legt fest, wie oft gewürfelt werden soll (Anzahl `N`) und welche Zahl als Treffer gezählt werden soll (Zahl `Z`).
  (*Benutzereingabe* mittels Befehl `input`.)
- Entsprechend wird `N`-mal eine Zufallszahl zwischen 1 und 6 generiert (Befehl `randi`) – das simuliert einen Wurf mit einem Spielwürfel.
  (Eine *Schleife*.)
- Bei jedem Wurf wird geschaut, ob die ausgewählte Zahl `Z` gewürfelt wurde. Wenn ja, wird die Anzahl der Treffer um 1 erhöht.
  (Eine *Verzweigung*.)

Beispiel: Programm Wuerfel.m

```
clc; clear all; close all;
N = input('Wie oft soll gewuerfelt werden? ');
disp('Welche Zahl soll als Treffer gezaehlt werden?')
Z = input('Gib eine Zahl zwischen 1 und 6 ein: ');

Zaehler = 0; % Zaehler fuer Z-Treffer
WurfNr = 1;
while WurfNr<=N
    % Eine Zufallszahl zwischen 1 und 6 wuerfeln.
    gewuerfelt = randi([1,6]);
    if gewuerfelt==Z % Wurde die Zahl Z gewuerfelt?
        Zaehler=Zaehler+1; % Dann erhoehe Zaehler um 1.
    end
    WurfNr = WurfNr+1; % Gehe zum naechsten Wurf.
end

disp(['Es wurde ',num2str(N),'-mal gewuerfelt.']);
disp(['Die Zahl ',num2str(Z),' wurde '])
disp([num2str(Zaehler),'-mal gewuerfelt.']);
```

Zusammenfassung

- Ablaufkonstrukte in (fast) allen Programmiersprachen: *Verzweigungen*, *Schleifen*, *Funktionen*.
- Verzweigungen: Einseitige Auswahl (`if–end`) und zweiseitige Auswahl (`if–else–end`).

  Die Bedingung in einer Verzweigung ist ein *logischer Ausdrück*, d.h., eine Behauptung, die entweder *wahr* oder *falsch* sein kann.
- *Schleifen* helfen die Vorgänge zu automatisieren. Wir haben eine *Wiederholschleife* (`while`-Schleife) kennengelernt.
- Schleifen und Verzweigungen können (fast) beliebig miteinander kombiniert werden.

MINT-Kolleg online unter

www.mint-kolleg.de

Baden-Württemberg

MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

Dem Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg (Studienmodelle individueller Geschwindigkeit) und dem Bundesministerium für Bildung und Forschung (Qualitätspakt Lehre, FKZ 01PL16018A, 01PL16018B) danken wir herzlich für die finanzielle Unterstützung dieses Projekts.