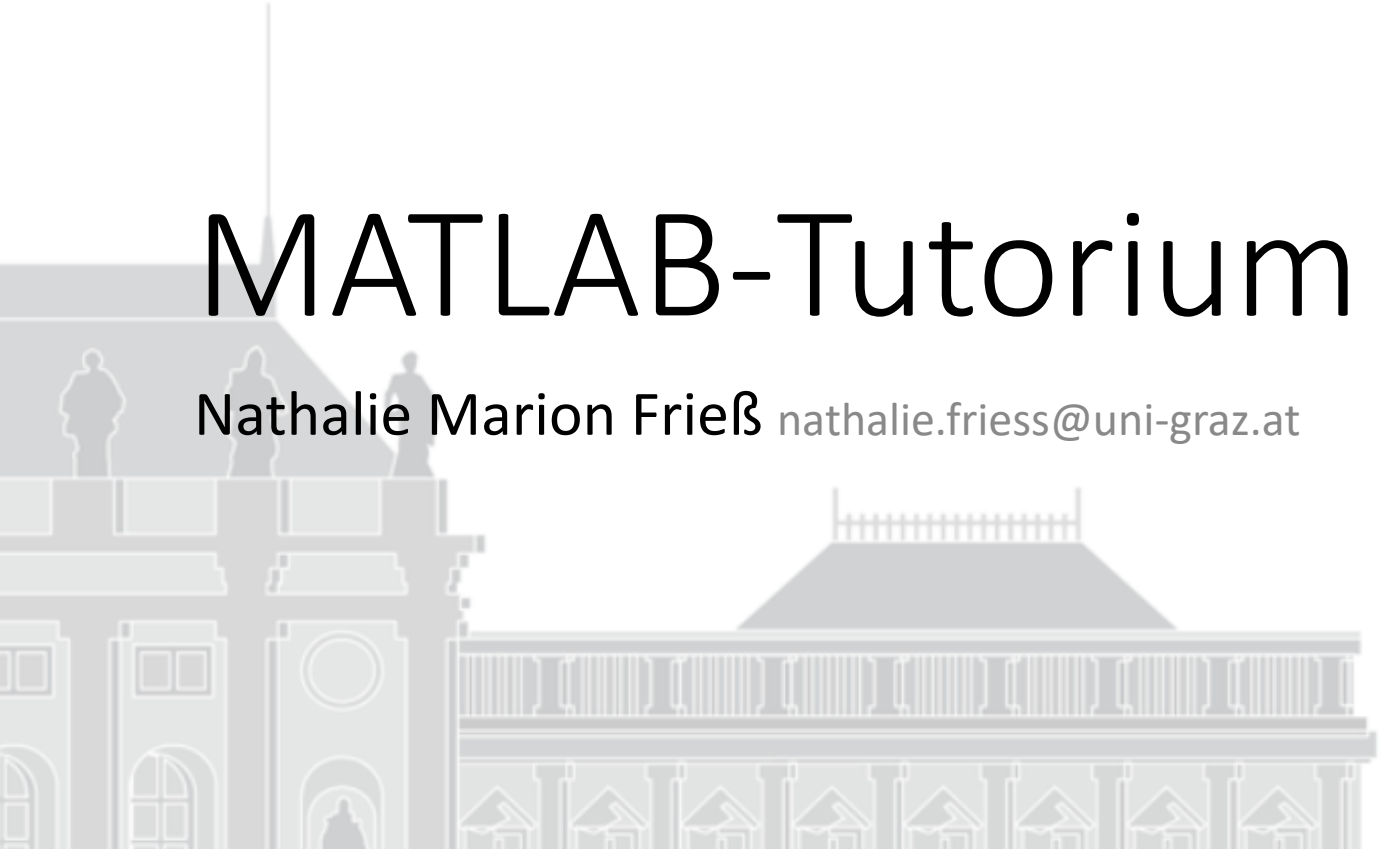
KARL-FRANZENS-UNIVERSITÄT GRAZ
UNIVERSITY OF GRAZ



MATLAB-Tutorium

Nathalie Marion Frieß nathalie.friess@uni-graz.at

Inhalt

- Einführung
- Das erste Programm
- Datentypen und Variablen
- Funktionen
- Steuerkonstrukte
- Eigene Funktionen
- Algorithmen
- Plots
- Flussdiagramme

Ziele

- Umgang mit dem Programm MATLAB
- Lösen praktischer Probleme aus OR/ProdLog
- Übersetzung in die Sprache des Computers
- Hilfestellung
- Speziell für Einsteiger

Was ist MATLAB?

- **MAT**rix **LAB**oratory
- Skriptsprache
- Mathematische Problemlösungen
- Matrizenrechnung
- Graphische Darstellungen mathematischer Funktionen
- Implementierung sehr einfach

Wieso MATLAB?

- Weite Verbreitung
- Hochschulen und Industrie
- Viele Anwendungen in quantitativer Datenauswertung
- Sinnvolle Zusatzqualifikation für quantitativ-orientierte Studierende
- Einfache Anwendbarkeit Internet-Hilfe

Vor- und Nachteile

+	-
Im Bereich der Numerik sehr schnell	Proprietär - kein Open Source Produkt
Mathematische Probleme sind bereits vielfach implementiert	Hohe Lizenzkosten
Sehr viele Online Tutorials und Dokumentationen	Freie Alternativen (z.B. Python, etc.)
Verfügbar für Windows, Linux und Mac OS	

Zugang

- UNI-IT Arbeitsplätzen lokal vorinstalliert
- Von zu Hause: Zugriff über Applikationsserver
  - Applikationsserver
  - Login mit UGO-Benutzerdaten (bzedvz\Benutzername)
  - Matlab: Matlab R2017a

Tipps & Tricks

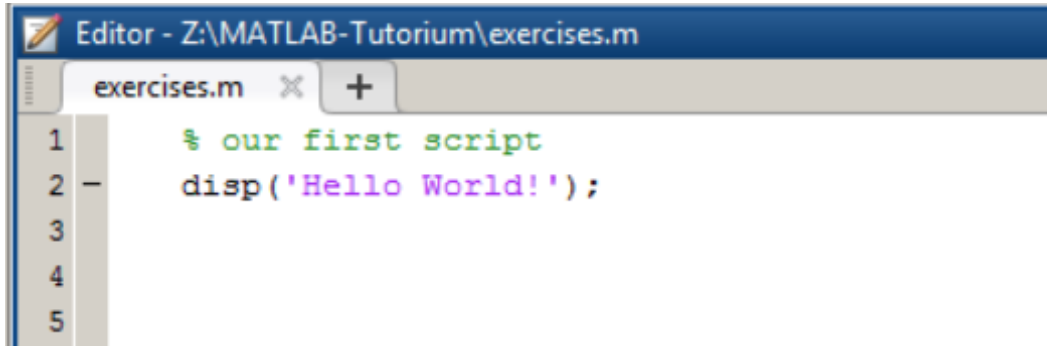
- Halte deinen Code sauber!
- KISS - Keep it short and simple
- DRY - Don’t repeat yourself - Copy & Paste verboten
- Englischer Code
- Sprechende Variablennamen verwenden
- Google ist dein Freund
- **R**ead**T**he**F**ine**M**anual: https://de.mathworks.com/help/
- Zerlege dein Problem in kleinere Teilprobleme

Wichtige Befehle

Befehl → Auswirkung

- help → Aufruf der Hilfe
- help Befehl → Aufruf der Hilfe zu einem Befehl
- clc → Löscht Inhalt des Command Windows
- clear all → Löscht alle angelegten Variablen
- who → Auflisten aller verwendeten Variablen

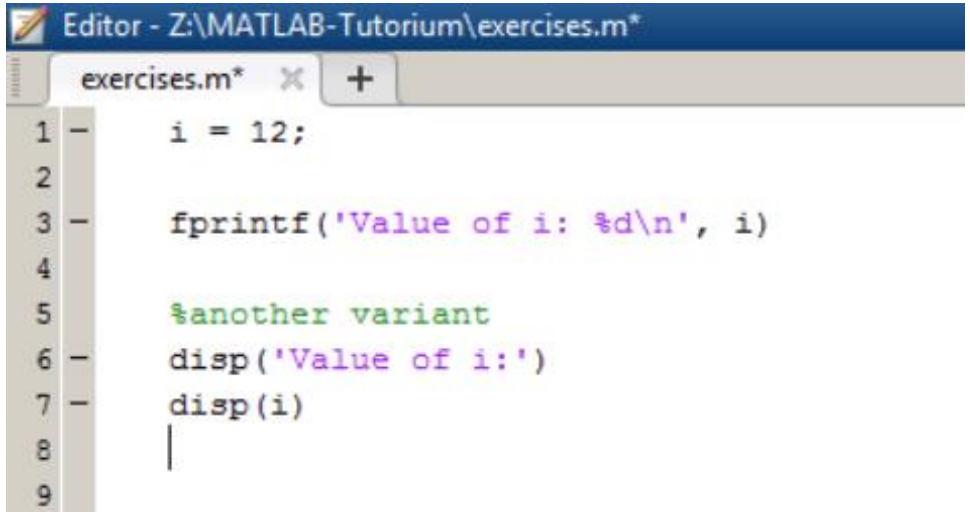
Das erste Programm



Drücke **F5** um das Programm zu starten

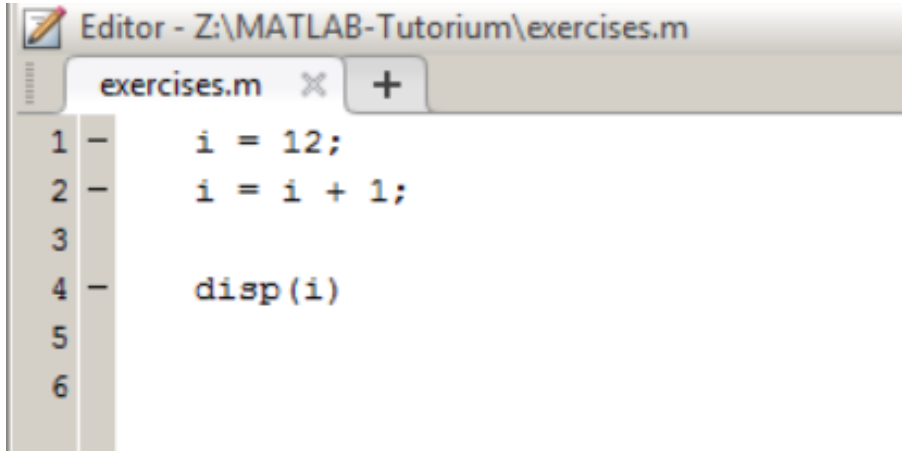
Datentypen und Variablen

- Deklaration durch erstmaliger Verwendung (Initialisierung)
- Dynamische Datentypen
  - Eine Variable ist ein Wert, gebunden an einen Namen
  - Der Wert hat einen Typ, aber die Variable an sich nicht



Datentypen und Variablen

Verwendung einer Variable auf beiden Seiten der Zuweisung



Achtung: Dieses Beispiel ist kein logischer Ausdruck, sondern eine Zuweisung. Der logische Ausdruck wäre natürlich immer *Falsch*!

Datentypen und Variablen

Benennung von Variablen

1. Variablennamen beginnen mit einem Buchstaben
2. Maximale Länge: 63 Zeichen
3. Matlab ist “case sensitive”: A unterscheidet sich von a
4. Vermeide i, j, pi und built-in Funktionsnamen wie length, char, size, …
5. Sprechende Variablennamen verwenden

Wichtige Datentypen

- Integer
- Float
- Strings
- Boolean
- Vektoren
- Matrizen
- etc.

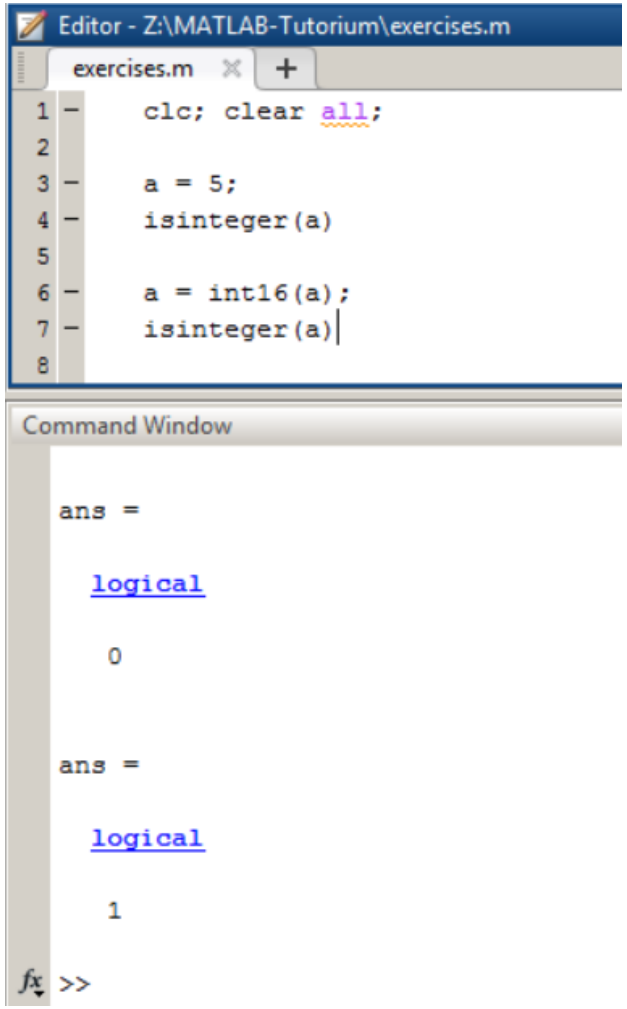
Wichtige Datentypen

- Integer

MATLAB speichert Zahlen automatisch

als Kommazahlen ab →

https://de.mathworks.com/help/matlab/matlab_prog/integers.html

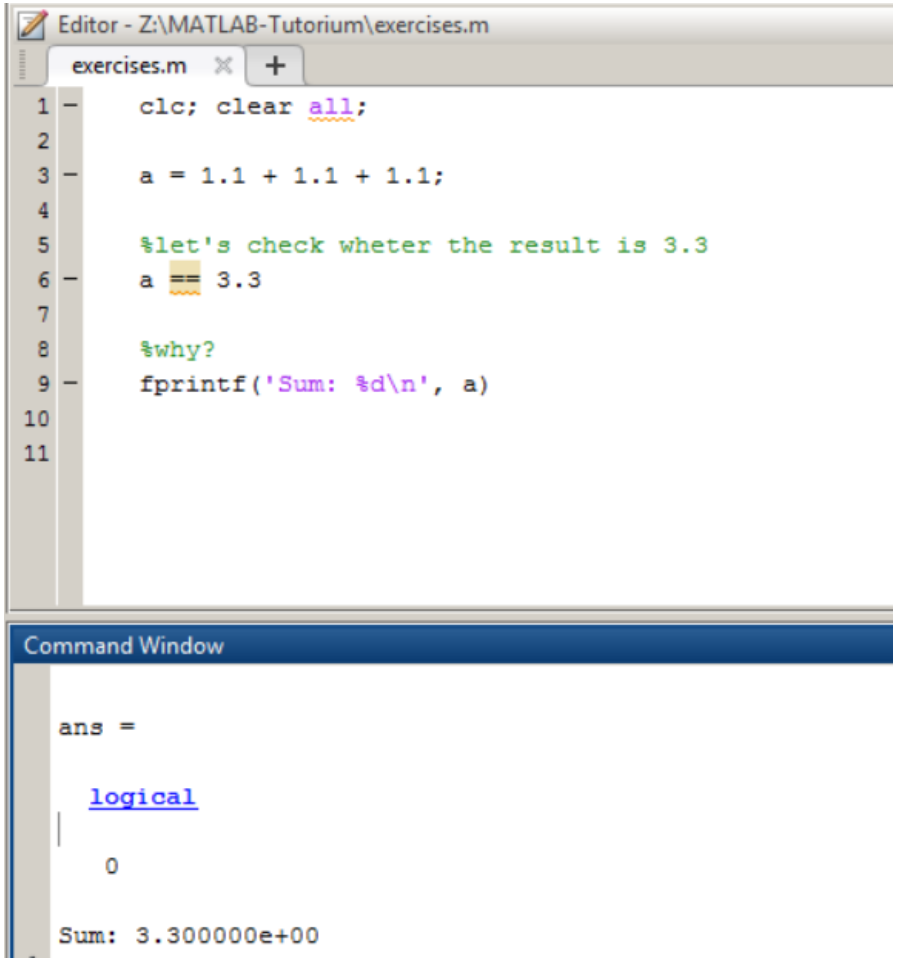


Wichtige Datentypen

- Float

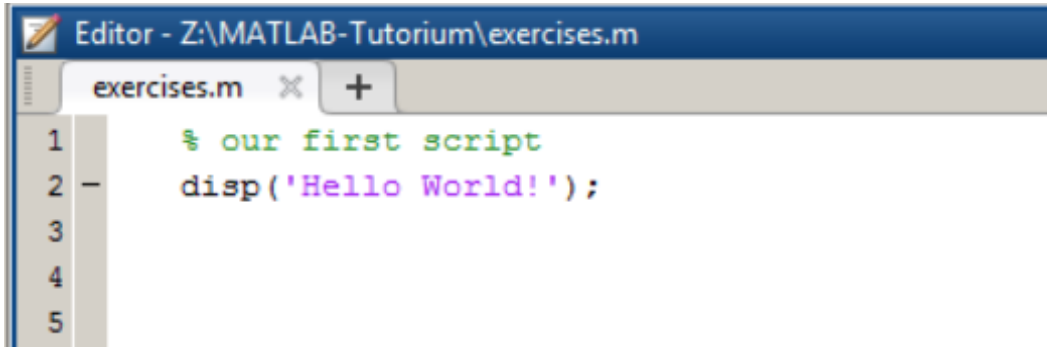
Häufige Fehlerquelle:

1.1 + 1.1 + 1.1 ≠ 3.3



Wichtige Datentypen

- Strings

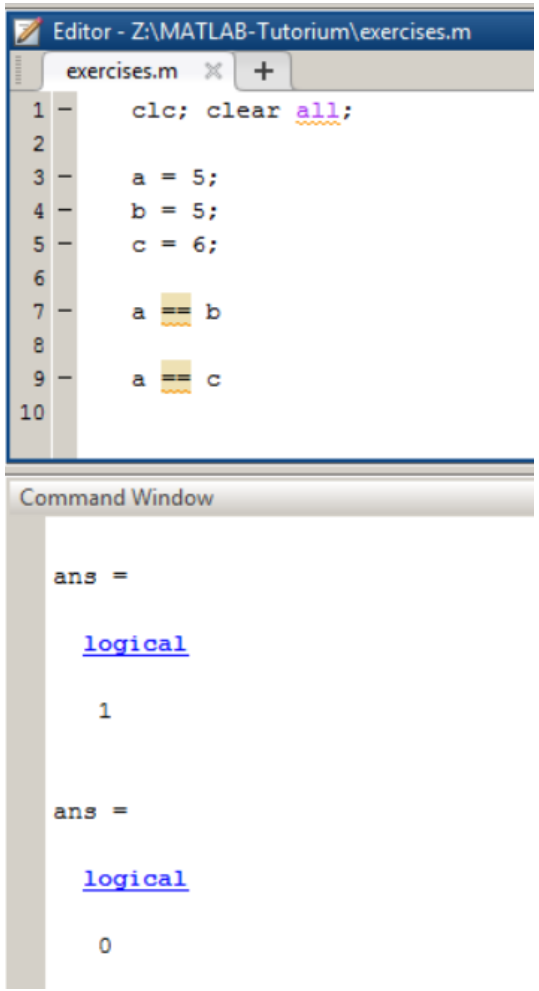


Wichtige Datentypen

- Boolean

Wahr → 1

Falsch → 0



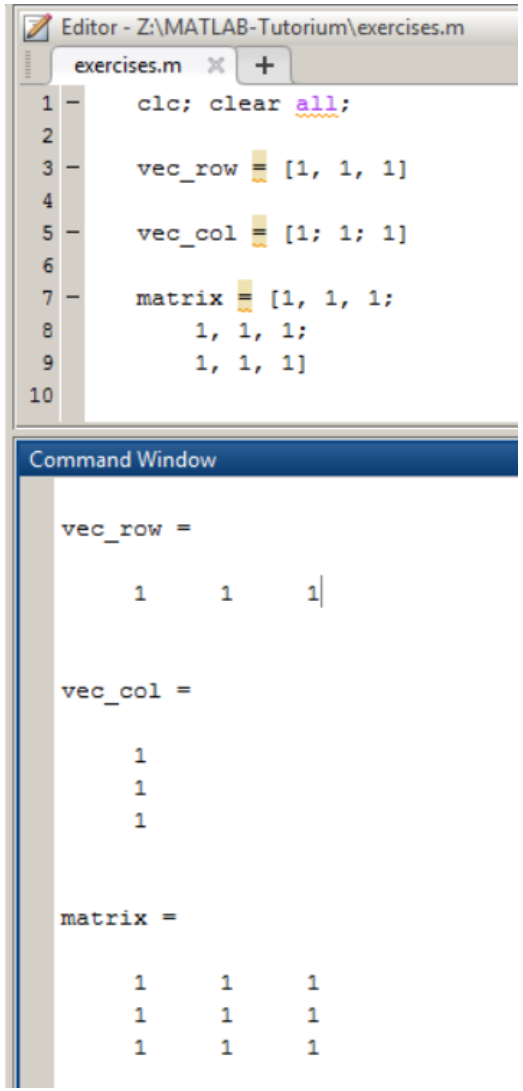
Wichtige Datentypen

- Vektoren und Matrizen

Zeilenvektoren [1, m]

Spaltenvektoren [n, 1]

Matrizen [n, m]



Wichtige Datentypen

- Vektoren und Matrizen

Zugreifen auf Elemente

Editor - Z:\MATLAB-Tutorium\exercises.m

exercises.m

```
1  clc; clear all;
2
3  matrix = [1, 2, 3;
4      4, 5, 6;
5      7, 8, 9]
6
7  %take the value of the second element in the first row and add 1
8  new_value = matrix(1,2)+ 1
9
10 %change the value of the second element in the third row
11 matrix(3,2) = 3
12
13 %save second row
14 second_row = matrix(2,:)
15
16 %save every second col of third row
17 vec = matrix(3,1:2:3)
18
```

Datentypen und Variablen

- Data Type Conversion: num2str(), str2num()

Editor - Z:\MATLAB-Tutorium\exercises.m*

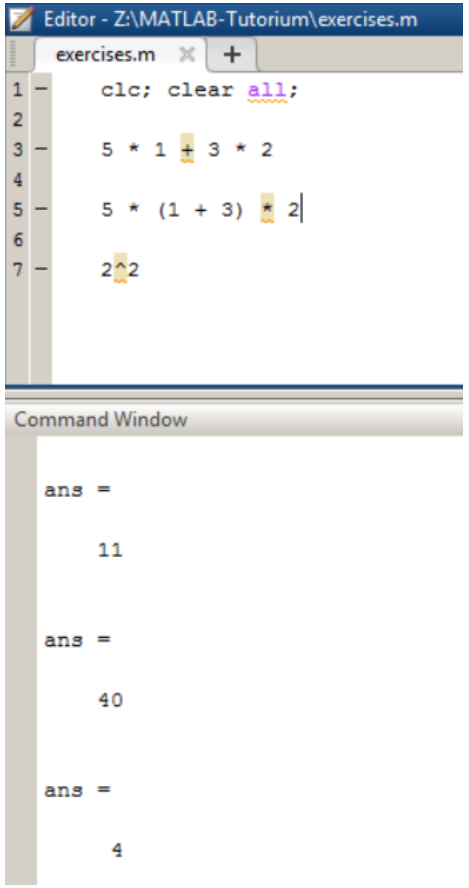
exercises.m*

```
1 -    clc; clear all;
2
3 -    myvar = '42';
4
5 -    mynumber = myvar + 0.5;
6 -    disp(mynumber)
7      %oh crap, sth wrong
8
9 -    mynumber = str2num(myvar) + 0.5;
10 -   disp(mynumber)
11     %that worked
```

Operatoren

Symbole die eine bestimmte Operation bewirken

+, -, *, /, ^



Operatoren

Boolsche Operatoren

==, ~= → ACHTUNG! Ein = ist eine Zuweisung, ein == ist ein Vergleich!

>, <

>=, <=

&&, ||

Übung 1

- Berechne 5% von 117.35
- Summiere die Zahlen von 1 bis 5 und nimm die 4. Potenz
- Berechne acht viertel gebrochen durch drei Achtel

Übung 2

- Weise der Variable *matrix* eine Matrix der Dimension 2x2 zu, die nur Einser enthält
- Weise der Variable *vektor* einen 5-elementigen Zeilenvektor zu, der nur Nullen enthält
- Erstelle einen 60-elementrigen Vektor, welcher ausschließlich aus 4en besteht
- Erstelle eine beliebige 2x2 Matrix deren Element (2,2) 5 ist
- Erstelle einen Vektor *vec* mit 8 Elementen. Weise der Variable *v* das 6. Element von *vec* zu. Ersetze das 3. Element von *vec* durch 90

Übung 3

- Erstelle 3 Variablen, welche jeweils einen von dir gewählten String enthalten
  - z.B. ’Hello World’ auf Variable string_one speichern, usw.
- Erstelle die Variable is_same_same, welche den Wert **true** erhalten soll, wenn:
  - Variante A: Alle drei Variablen den gleichen String enthalten
  - Variante B: Zumindest zwei der drei den gleichen String enthalten
  - Variante C: Genau zwei der drei den gleichen String enthalten

Aufrufen von Funktionen

- Eine Funktion nimmt bestimmte Parameter entgegen und liefert bestimmte Rückgabewerte
- Eine Funktion kann man sich als kleines Unterprogramm bzw. als Blackbox vorstellen
  1. Die Funktion nimmt Werte als Parameter entgegen
  2. Die Funktion macht irgendetwas
  3. Sobald die Funktion fertig ist, geht mein Code wieder weiter. Ich kann mir den Output der Funktion auf meine eigenen Variablen speichern.
  4. Eine Funktion hat immer eine bestimmte Aufgabe. Die Aufgabe der Funktion size(x) ist z.B., den Wert x zu übernehmen, davon die Länge zu bestimmen, und diese Länge an das Hauptprogramm zurückzugeben.

Die wichtigsten built-in Funktionen

Funktion	Verwendung
sqrt()	Quadratwurzel
zeros()	Matrix mit 0en erzeugen
ones()	Matrix mit 1en erzeugen
rand()	Zufallszahl erzeugen
abs()	Absolutbetrag
sum()	Summe aller Werte eines Vektors
prod()	Produkt aller Werte eines Vektors
min/max()	Minimum/Maximum finden
sort()	Vektor sortieren

Die wichtigsten built-in Funktionen

Funktion	Verwendung
sortrows()	Sortiert Zeilen einer Matrix
length()	Länge eines Vektors
size()	Dimension einer Matrix
inv()	Gibt Inverse einer Matrix
exp()	Exponentialfunktion
log()	Logarithmusfunktion
input()	Fragt den User nach Eingabe
fprintf()	Gibt formatierten Text aus
disp()	Gibt Wert einer Variable aus

Übung 4

- Erstelle einen Vektor mit 10 Zufallszahlen zwischen [2,5]; ziehe von jedem einzelnen Element die Wurzel und speichere das Ergebnis wiederum auf den ursprünglichen Vektor; sortiere den Vektor nun aufsteigend
- Erzeuge einen 10 elementrigen Zufallszahlen-Vektor und finde das Maximum
- Erstelle einen beliebigen Vektor und überschreibe das Maximum mit der Zahl -1
- Frage den User nach Eingabe einer Zahl und addiere 2

Übung 5

- Gegeben ist folgende Input Matrix:

1	2	3	4	5
5	4	21	15	10
10	5	30	50	52

- Nun sortiere diese Matrix nach der 3. Zeile, sodass das Ergebnis ist:

2	1	3	4	5
4	5	21	15	10
5	10	30	50	52

Übung 6

- Gegeben ist folgende Input Matrix:

2	1	3	4	5
4	5	21	15	10
5	10	30	50	52

- Suche nun nach jener Spalte mit dem größten Wert in der 2. Zeile und lösche diese Spalte → Ergebnis:

2	1	4	5
4	5	15	10
5	10	50	52

Übung 7

- Frage den User um Eingabe einer Zahl
- Erstelle einen Vektor, welcher alle Zahlen von 1 bis zur eingegebenen Zahl enthält
- Überprüfe zur Sicherheit, ob dein erstellter Vektor tatsächlich genügend groß ist (ob die Länge deines Vektors der eingegebenen Zahl entspricht)

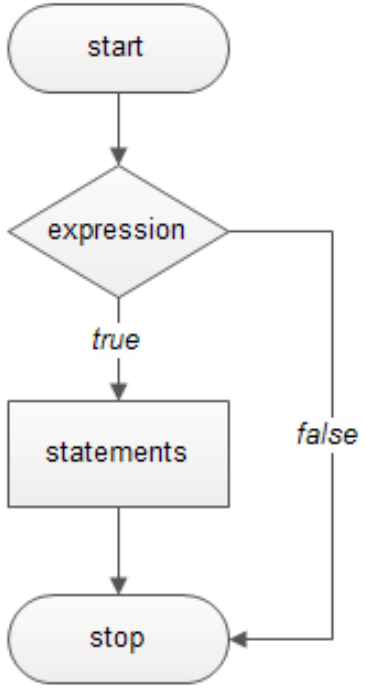
Steuerkonstrukte

- Bisher: Durchlauf eines Files von oben bis unten
- Jetzt:
  - Manche Blöcke mehrmals durchlaufen mit unterschiedlichen Parametern
  - Bestimmte Zeilen nur unter bestimmten Bedingungen ausführen
- Steuerkonstrukte regeln in welcher Reihenfolge ein File durchlaufen wird, was ausgeführt und ausgelassen wird
- Die wichtigsten Steuerkonstrukte:
  - If-Bedingungen
  - For-Schleifen
  - While-Schleifen

If-Bedingungen

Ein if-Block hat folgende Form:

```
if expression
    statements
elseif expression
    statements
elseif expression
    statements
else
    statements
end
```



Der Block beginnt mit *if* und endet mit *end*. Beliebig viele *elseif* und maximal ein *else* sind optional.

Übung 8

- Wenn x kleiner 5 ist multipliziere mit 1/100, wenn x gleich 5 ist multipliziere mit 1/150 und sonst multipliziere mit 1/10
- Wenn x ein Vektor ist, fahre fort, sonst gib Fehlermeldung aus
- Erzeuge eine Zufallszahl. Wenn diese < 0.5 ist, dann erzeuge eine weitere Zufallszahl. Falls diese nochmals < 0.5 gib den Text “2x kleiner“ aus. Falls diese wiederum größer ist, gib aus: “1x kleiner, 1x groesser“

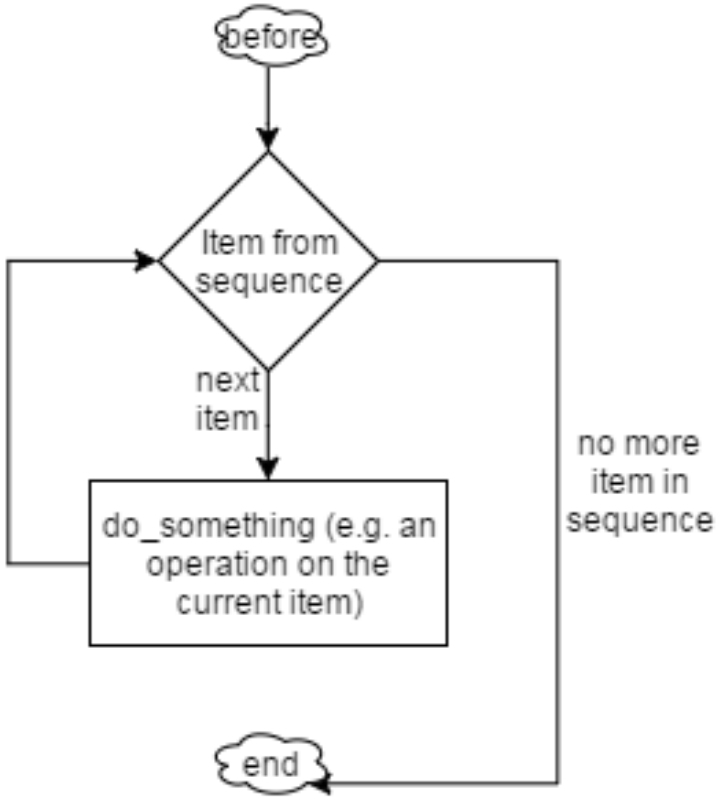
for-Schleifen

Ein for-Block hat folgende Form:

Editor - Z:\MATLAB-Tutorium\forloop.m*

forloop.m*

```
1      %SYNTAX
2  -   for index = start:ende
3  -       statements
4  -   end
5
6  -   for index = values
7  -       statements
8  -   end
9
```



for-Schleifen

Eine Schleife kann früher verlassen werden, wenn eine gewisse Bedingung eintritt:

- Der Befehl dazu lautet *break* und wird meist zusammen mit einer If-Abfrage verwendet
- Wird *break* verwendet, nennt man die Schleife bedingte For-Schleife

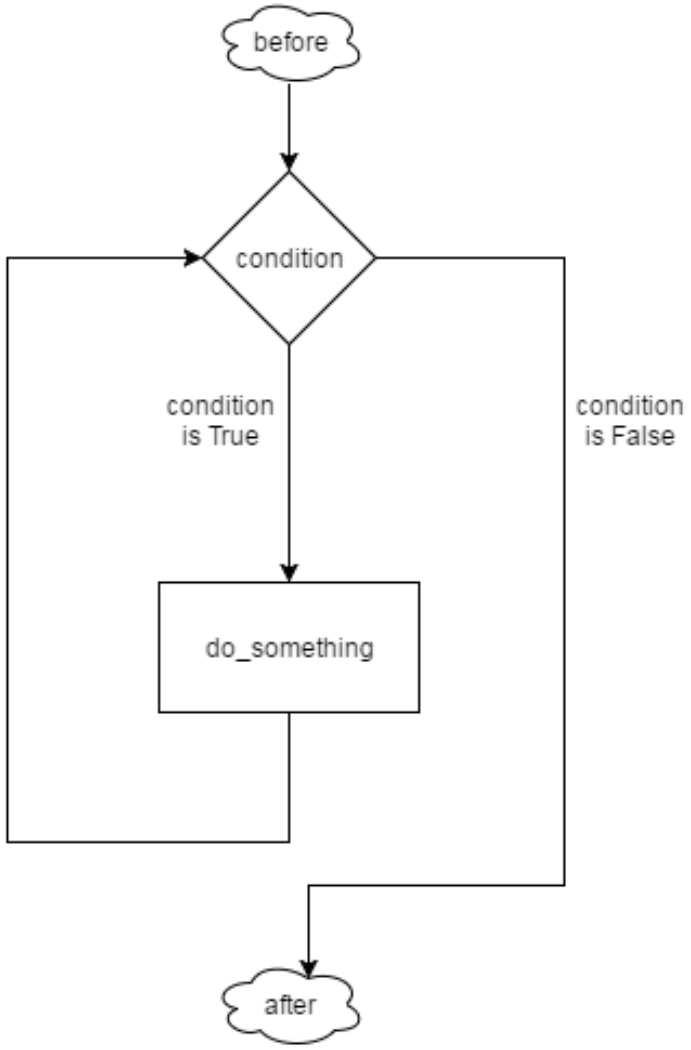
while-Schleifen

Ein while-Block hat folgende Form:

Editor - Z:\MATLAB-Tutorium\whileloop.m*

whileloop.m*

```
1     %SYNTAX
2
3 -   while expression
4 -       statements
5 -   end
6
```



while-Schleifen

Achtung vor ENDLOS-SCHLEIFEN!

Ist die Anweisung einer Schleife so, dass sie nicht mehr verlassen werden kann, rechnet der Computer immer weiter!

Unterbrechung der Berechnung mit Strg+c im Command Window

Übung 9

- Erste Variante ohne Schleifen:
  - Lass dir vom User 2 Zahlen eingeben.
  - Du berechnest das Produkt der beiden Zahlen und gibst das Ergebnis aus
- Nun mittels for Schleife:
  - Zuerst fragst du, wieviele Zahlen er/sie eingeben möchte
  - Danach liest du alle ein und gibst das Produkt aller Zahlen aus
- Nun mittels while Schleife:
  - Lass dir solange Zahlen eingeben, bis der User nichts mehr eingibt und einfach auf ENTER drückt.
  - Gibt am Schluss das Produkt aus

Übung 10

- Gegeben ist folgende Input Matrix:

job	1	2	3	4	5
processing time	4	5	21	15	10
due date	5	10	30	50	52

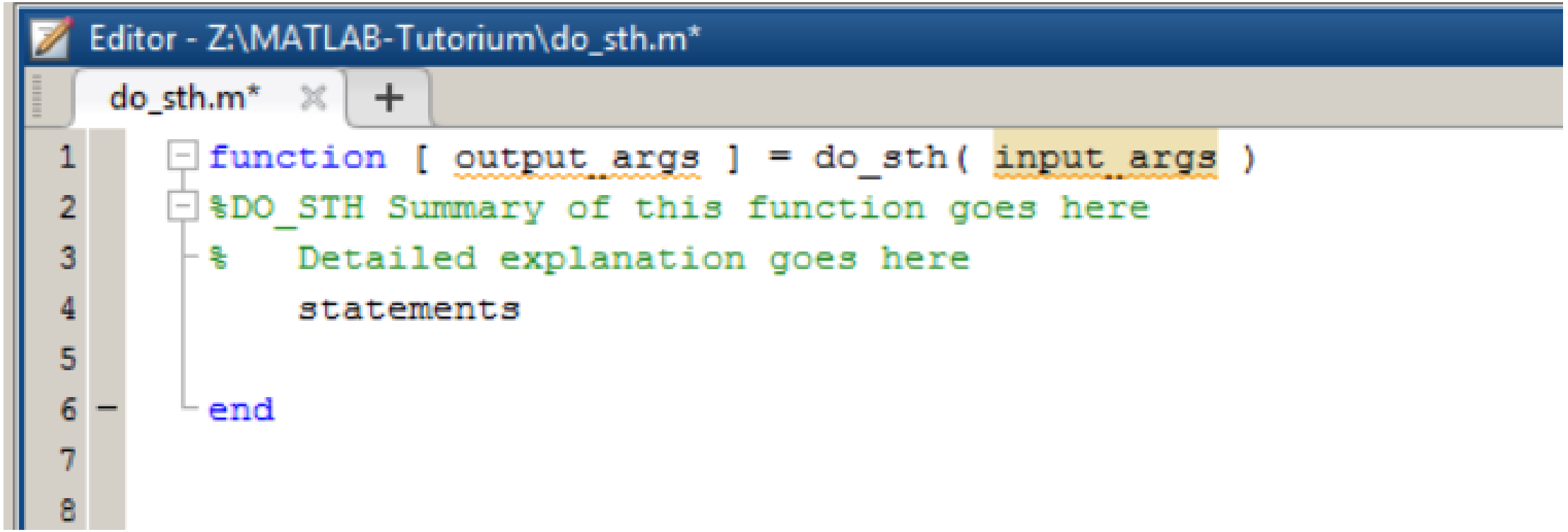
- Erzeuge nun eine neue Matrix auf Basis der Input Matrix, welche eine 4. Zeile *completion_time* hat, wobei hier jeweils die *completion_time* der vorherigen Spalte + die processing time der aktuellen Spalte stehen soll
- Formal: $M_{4,j} = M_{4,j-1} + M_{2,j}$ und $M_{4,1} = M_{2,1}$
- Ergebnisvektor $M_{4,:}$ sollte sein: 4 9 30 45 55

Eigene Funktionen

- Warum?
  - Code öfters wieder verwenden, aber nur 1x schreiben
  - Um das Programm übersichtlich zu halten
  - Das Problem in kleinere Teilprobleme zu zerlegen
- Bottom-up vs. top-down programming
- Wann benötige ich Funktion?
  - Ein gewisser Teil vom Code lässt sich verbal kurz zusammenfassen.

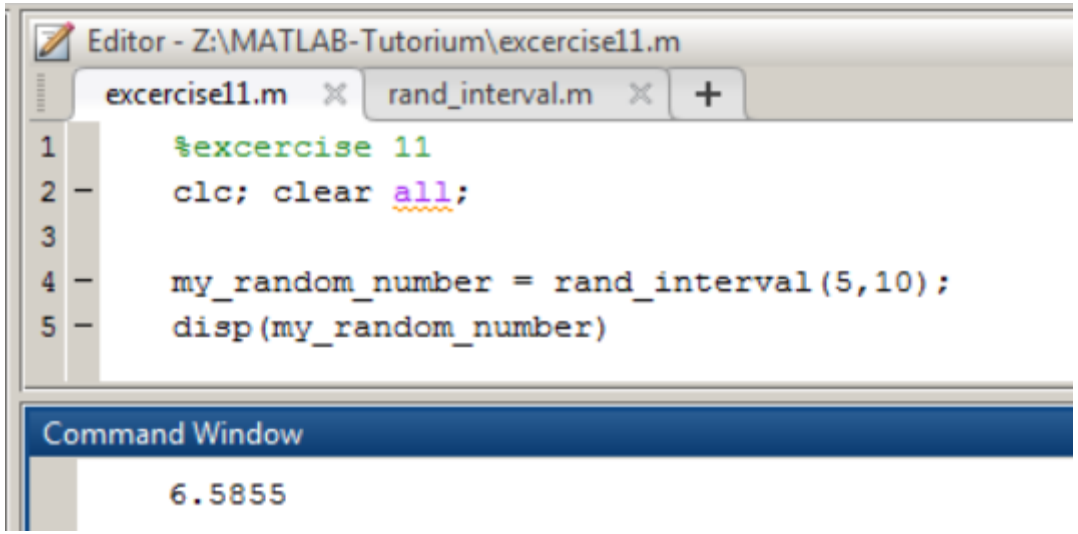
  Beispiel: Die letzten 10 Zeilen wurden benötigt, um eine Matrix zu initialisieren → in Funktion initializeMatrix() auslagern
  - Code hat mehr Zeilen als am Bildschirm Platz → Funktion
  - Ich würde Copy & Paste benötigen → Funktion

Eigene Funktionen



Übung 11

- Erzeuge eine Funktion *rand_interval*, welche eine Zufallszahl zurück gibt. Diese Zufallszahl soll zwischen den Inputparametern lower_bound und upper_bound sein.



Obiger Aufruf der Funktion soll eine Zahl zwischen 5 und 10 ausgeben.

Algorithmen

- Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder Klasse von Problemen
- Unterschied zwischen Algorithmus und Solver:
  - Solver verwendet Algorithmus zur Lösung eines mathematischen Problems (zB ILP)
- Standard-Algorithmen sind bereits in Programmiersprache als Funktion vorhanden:
  - Finden des Maximums
  - Sortieralgorithmus
  - etc.

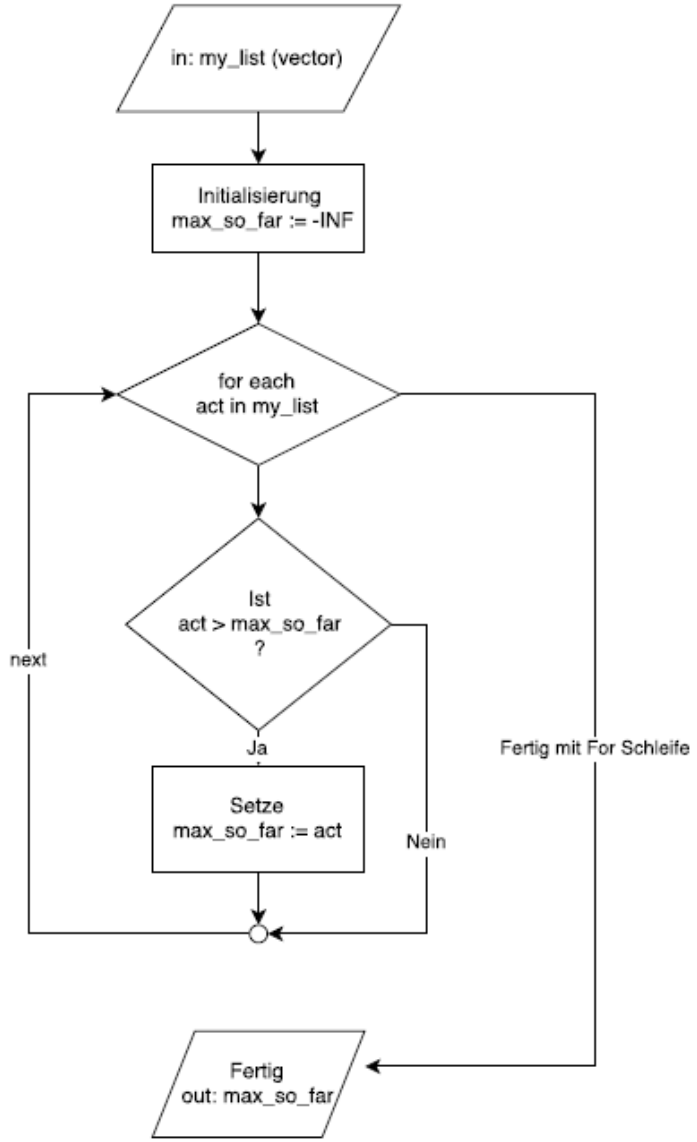
Algorithmen

Finden des Maximums:

```
4      %initialize
5 -    my_list = [3, 2, 6, 76, 45];
6 -    max_so_far = -inf;
7
8
9      %iterate through all values in list
10 -   for elem_nr = 1:length(my_list)
11 -       act = my_list(elem_nr);
12 -       if act > max_so_far
13 -           max_so_far = act;
14 -       end
15 -   end
16 -   disp(max_so_far)
```

Command Window

```
    76
```



Algorithmen

Bubble Sort:

https://en.wikipedia.org/wiki/Bubble_sort

Editor - bubblesort.m

bubblesort.m

```
%bubblesort
clc; clear all;

my_list = [54, 3, 5, 2, 34];

swapped = true;

while swapped == true %as long as sth changed in last round
    swapped = false; %set swapp false and see if sth changes
    for index = 1:length(my_list)-1 %why -1? see yourself whats happening without it!
        %save the two elements we're comparing
        element = my_list(index);
        next_element = my_list(index+1);
        if element > next_element
            %swap if necessary
            my_list(index) = next_element;
            my_list(index+1) = element;
            swapped = true;
        end
    end
end
disp(my_list)
```

Variables - my_list

Command Window

```
     2     3     5    34    54
```

Algorithmen

- Gegeben: Jobliste mit Prozesszeit und Due Date
- Output: Alle Jobs, die rechtzeitig fertig gestellt werden und zugehöriger Schedule

Moore in a Nutshell:

- Sortiere JOBS nach EDD
- $\forall job\ k\ \in JOBS_{EDD}$
  - Setze Job k auf Liste L (vorläufiger Schedule)
  - Sollte Job k late sein (cj > dj):
    - Lösche Job mit größtem pj aus L
- L ist fertiger Schedule (alle Jobs nicht in L sind late)

Plots

- Funktion: plot(x,y)
- x und y sind dabei Vektoren der gleichen Länge und enthalten die x- und y-Koordinaten der zu zeichnenden Datenpunkte
- BSP: Zeichnen der Wurzelfunktion im Interval [0,5]

  >> x = 0:0.5:5; y = sqrt(x); plot(x,y)

  Anzahl der gezeichneten Datenpunkte vorgeben → Vektor x mit linspace(erster Wert, letzter Wert, Anzahl der Werte*) erzeugen

  *Anzahl der Werte nicht angegeben → automatisch 100

Plots

Veränderung des Aussehens → dritter Parameter *LineSpec:*

- Farbe → y,m,c,r,g,b,w,k
- Markierung am Datenpunkt → o,+,*,.,x,s,d,^,v,>,<,p,h
- Linienart → -,--,-.,:

BSP: rot, an den Datenpunkten Stern, strichlierte Linie

>> plot(x,y,'r*–')

Plots

Beschriftung →

- x-Achse wird mit Hilfe von xlabel beschriftet (>> xlabel('x-Achse'))
- y-Achse wird mit Hilfe von ylabel beschriftet (>> ylabel('y-Achse'))
- Titel mit Hilfe von title (>> title('Grafik'))
- Funktion mit Hilfe von text (>> text(1,1,'Funktionsname'))

  Zusätzlich muss Ausrichtung angegeben werden (Koordinaten des Startpunkts, an dem Text beginnt)

Plots

- Gitternetz: Befehl *grid*
- Mehrere Funktionen:
  - Verwendung von *hold*

    BSP: >> plot(x,y); hold; plot(x,z);
  - Funktionen hintereinander eingeben

    BSP: >> plot(x,y,'r*-',x,z,'b')
- Grafikfenster in mehrere Teilfenster aufteilen: subplot(m,n,p)
  - m…Anzahl der Grafiken übereinander
  - n...Anzahl der Grafiken nebeneinander
  - p…aktuelle Grafik

Übung 12

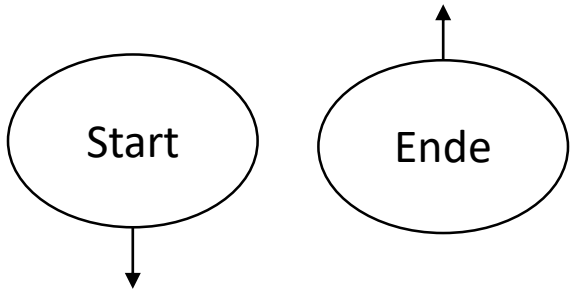
- Zeichne die Funktionen y=sin(x) und y=cos(x) mit einem Gitternetz in ein Koordinatensystem
- Zeichne die Funktionen y=sin(x), y=cos(x), y=-sin(x) und y=-cos(x) in getrennte Teilgrafiken

Flussdiagramme

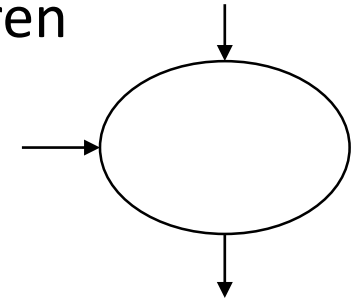
- Beschreibung von Programmabläufen um die Vorgehensweise eines Programms zu veranschaulichen
- bestehen aus unterschiedlich geformten Elementen, die mit Pfeilen verbunden sind
- Pfeilrichtungen geben die Verarbeitungsreihenfolge vor
- jedes Element beschreibt einen einfachen Verarbeitungsschritt
- kann unter anderem in PowerPoint gezeichnet werden

Flussdiagramme

- Anfang und Ende werden durch einen kleinen Kreis dargestellt um Pfeile zu vermieden, die aus dem Nichts auftauchen bzw. dorthin münden



- Punkte, an denen mehrere Pfeile zusammengeführt werden, werden ebenfalls mit einem kleinen, leeren Kreis markiert um die Rolle einer Einmündung ausdrücklich zu dokumentieren

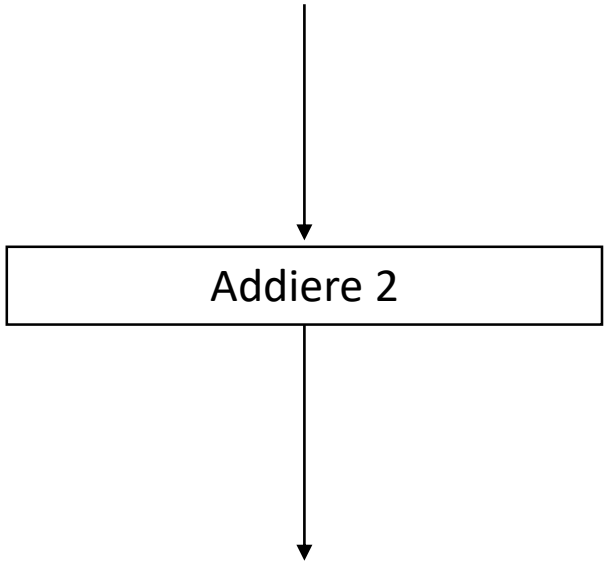


Flussdiagramme

- Anweisungen werden in Rechtecke geschrieben

- Sequenzen werden als Pfeile dargestellt

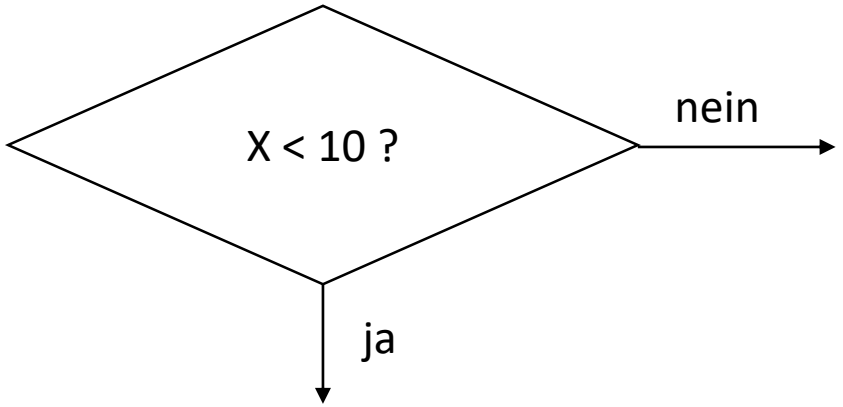
  Sie stellen den Ablauf des Programms dar



Normalerweise verläuft der Hauptkontrollfluss von oben nach unten

Flussdiagramme

- Verzweigungen sind an Bedingungen geknüpft (kommt bei if und while vor)
- Eine Bedingung wird in eine Raute gesetzt, aus der 2 Pfeile herausführen:
  - einer der Ausgänge (meistens nach unten gerichtet) ist mit *Ja* beschriftet und wird weiterverfolgt, wenn die Bedingung erfüllt ist
  - der andere Ausgang (meistens zur Seite gerichtet) ist mit *Nein* beschriftet und wird andernfalls weiterverfolgt



Flussdiagramme