Datentypen und Fallunterscheidungen

Stand: 22.08.2016

Benötigte Dateien:
[phWerte.mat]
[katzen.mat]
[if_demo.mat]

A) FALLUNTERSCHEIDUNGEN

Bisher entsprachen alle Programme im Kurs einem einfachen Kochrezept: Einzelne Schritte wurden der Reihe nach abgearbeitet (linearer Algorithmus). Jetzt gehen wir einen Schritt weiter und führen Alternativen ein, wie an einer bestimmten Stelle der weitere Programmablauf fortgesetzt werden soll. (Z.B. "Wenn Sie Heidelbeermuffins backen wollen, fügen Sie dem Muffinteig Heidelbeeren hinzu, alternativ können Sie auch Himbeeren verwenden.") Diese Verzweigungen im Programmfluss nennt man Fallunterscheidungen.
Fallunterscheidungen werden häufig verwendet, um für verschiedene Zahlenbereiche einer Variablen unterschiedliche Operationen vorzunehmen. Um die Zahlenbereiche voneinander zu trennen verwendet man die Vergleichsoperatoren

- < kleiner
- > größer
- == Test auf Gleichheit
- <= kleiner oder gleich
- >= größer oder gleich
- ~= ungleich

Die allgemeine Syntax für eine Fallunterscheidung lautet:

```
if bedingung
 befehle1
else
 befehle2
end
```

Z.B. könnte man mit positiven Zahlen anders umgehen wollen als mit negativen und Null:

```
a=19.7 % oder jede andere Zahl
if a>0 % fuer positive Zahlen
 b=2*a % setze b=2*a
else % fuer negative Zahlen und 0
 b=-2a % mache b positiv
end
```

Die Einrückung wird durch den Matlabeditor automatisch vorgenommen. Sie ist für die Ausführung des Programms nicht zwingend notwendig, erleichtert aber sehr die Lesbarkeit.
Der else-Teil der Fallunterscheidung kann auch weggelassen werden, wenn nur im Fall einer zutreffenden Bedingung etwas passieren soll und sonst nichts zu tun ist. Z.B.

```
a=-19.7 % oder jede andere Zahl
```

```
% a soll auf jeden Fall eine positive Zahl sein
if a<0 % fuer negative Zahlen
 a=-a % mache a positiv
end
```
Wenn mehr als zwei Bedingungen unterschieden werden, braucht man die Syntax
```
if (bedingung1)
 befehle1
elseif(bedingung2)
 befehle2
else
 befehle3
end
```
(Dabei kann es beliebig viele elseif-Teile geben).
Bedingungen können auch kombiniert werden:
```
bedingung1 && bedingung2 % "Und"-Verknuepfung, beide muessen gelten
bedingung1 || bedingung2 % "Oder"-Verknuepfung, eine von beide muss gelten
```
z.B.
```
if a>0 && b<0
 b=1;
 c=a;
elseif a<0 && b>0
 a=1;
 c=b;
else
 c=a*b
end
```

Aufgaben:

T3A1) Schreiben Sie eine Funktion, die eine Zahl als Eingabeparameter bekommt. Wenn diese Zahl negativ ist, wird sie auf dem Bildschirm ausgegeben, sonst passiert nichts.

T3A2) Schreiben Sie eine Funktion, die eine Matrix als Eingabeparameter erwartet. Wenn diese Matrix quadratisch ist, gibt die Funktion die Zahl 1 zurück, sonst die Zahl 0.

T3A3) Schreiben Sie eine Funktion, die eine Zahl als Eingabeparameter bekommt.

- Wenn dieser eine gerade Zahl ist, gibt die Funktion die Hälfte der Zahl zurück.
- Wenn der Eingabeparameter eine 0 ist, wird eine 1 zurückgegeben
- sonst wird die Zahl selber zurückgegeben.

T3A4) Schreiben Sie eine Funktion, die zwei Zahlen als Eingabeparameter bekommt und die Anzahl der positiven Eingabeparameter zurückgibt.

B) DATENTYPEN: ZAHLEN

Die wichtigsten Datentypen in Matlab haben Sie bereits kennengelernt: Skalare Werte, Vektoren und Matrizen, die aus Zahlen bestehen.

- Variable, die Zahlen (numeric) enthalten, werden im Matlab-Editor durch vier Kästchen symbolisiert
- Zahlen sind in Matlab normalerweise definiert als Fließkommazahlen mit doppelter Präzision (*double*).
- Es gibt aber z.B. auch ganze Zahlen (*int*),
- Welchen Datentyp man wählt, hängt davon ab, welche Art von Daten man damit darstellen möchte.

Fließkommazahlen (*double*):

- Fließkommazahlen können sehr viele Stellen hinter dem Komma haben, z.B. die Zahl PI.
- Wenn Sie sich *pi* anzeigen lassen, sehen Sie nur vier Nachkommastellen. Tatsächlich rechnet Matlab aber mit sehr viel mehr Stellen.
- Trotzdem muss man sich dessen bewusst sein, dass Computer grundsätzlich nur mit endlicher Genauigkeit (in Matlab festgelegt durch die Konstante *eps*) rechnen können, also die Nachkommastellen an einer bestimmten Stelle abgeschnitten werden.
- Auch gibt es eine Obergrenze des bearbeitbaren Zahlenbereichs (Konstante *realmax*). Oberhalb dieser Zahl gibt es für Matlab nur noch eine Zahl, mit der zuverlässig gerechnet werden kann: *inf* (infinity, also Unendlich).

Ganze Zahlen (int):

- Ganze Zahlen haben ebenfalls einen beschränkten Darstellungsbereich.
- Je nach dem, wie groß die Zahlen sind, die man mit einer ganzzahligen Variablen darstellen möchte, gibt es in Matlab verschiedene Typen, die unterschiedlich viel Speicherplatz brauchen: *int8*, *int16*, *int32* und*int64*. Dabei gibt die Zahl jeweils an, wie viele bit an Speicherplatz verwendet werden. Z.B. können mit mit einer Variable vom Typ *int8* 2^8=256 Zahlen dargestellt werden. Da sowohl positive als auch negative Zahlen und Null im Bereich von int8 enthalten sind reicht der darstellbare Bereich von -128 bis 127.
- Wenn man sicher ist, dass man nur positive Zahlen verwenden möchte, kann man auch *uint8*, *uint16*,*uint32* und *uint64* ("unsigned integer") verwenden. Für *uint8* geht der Zahlenbereich von 0 bis 255.

Zur Umwandlung des Typs einer Variablen benutzt man den Namen des gewünschten Datentyps. z.B. macht *uint16(a)* aus der Variable *a* eines beliebigen Datentyps eine durch 16 bit dargestellte ganze Zahl.

Aufgaben:

T3B1) Man kann mit dem Befehl *format* umstellen, wie viele Nachkommastellen einer Fließkommazahl gezeigt werden.

- Probieren Sie aus, wie *pi* und wie *100*pi* angezeigt werden nachdem sie jeweils einen der Folgenden Befehle angegeben haben:
  - *format long*
  - *format long e*
  - *format short e*
  - *format shortformat short g*
- Für welche Anwendungen ist welches Format am besten geeignet?

T3B2) Computer können nur einen begrenzten (wenn auch ziemlich großen) Zahlenbereich darstellen und verarbeiten.

- Schauen Sie sich die größte und die kleinste positive Zahl an, mit der Ihr Rechner mit Matlab zuverlässig rechnen kann: *realmax, eps*
- Was passiert, wenn Sie diesen Bereich über- bzw. unterschreiten? Probieren Sie z.B. *realmax*10, eps/2, realmax+1, 10-eps*

T3B3) Zwei besondere Konstanten in Matlab freuen einen meistens nicht, wenn man sie als Ergebnis einer Berechnung geliefert bekommt: *inf* und *nan*.

- Schauen Sie in der Hilfe nach, was diese bedeuten. Was ist der Unterschied zwischen ihnen?
- Versuchen Sie, die beiden Konstanten durch Rechenoperationen zu erzeugen.
- Erzeugen Sie zwei Matrizen, in denen einmal an einer Stelle *nan* und in der anderen an einer Stelle *inf* vorkommt, während alle anderen Matrixelemente normale Zahlen sind.
- Versuchen Sie mit beiden Matrizen zu rechnen. Was passiert?

T3B4) Es gibt Fälle, für die gebrochene Werte keinen Sinn ergeben, sondern ausschließlich ganze Zahlen (*integer*), z.B. wenn es um die Anzahl von Tieren geht. Insbesondere ist das auch der Fall, wenn Sie ein Element eines Vektors indizieren wollen.

- Stellen Sie sich vor, Sie haben einen Vektor v, der 100 Datenpunkte einer Messreihe enthält und wollen wissen, wie groß Ihr Messergebnis nach 1/3 der Zeit war. Probieren Sie aus, ob das so geht:

```
v=0.1:0.1:10;
L=length(v);
ind=L/3;
m=v(ind)
```

- Wie sieht die Antwort von Matlab im Vergleich aus, wenn Sie den Messpunkt nach 1/2 der Zeit betrachten wollen?

- Um die Fehlermeldung los zu werden, können Sie den Index explizit zu einer Integer-Variable machen:

```
v=0.1:0.1:10;
L=length(v);
ind=int16(L/3);
m=v(ind)
```

- Eine Alternative dazu ist, weiterhin mit dem Standard-Datentyp double zu arbeiten, aber die entsprechende Zahl zu runden:

```
v=0.1:0.1:10;
L=length(v);
ind_a=round(L/3);
m_a=v(ind_a)
```

- Allerdings kann man so eine Rundung leicht vergessen, insbesondere wenn mit den Zahlen weiter gerechnet wird. Sicherer ist deshalb, integer zu verwenden, wenn gebrochene Zahlen keinen Sinn machen. Probieren Sie aus:

```
ind=int16(L/3);
m=v(ind);
ind2=ind/2;
m2=v(ind2)
```

- versus

```
ind_a=round(L/3);
m=v(ind_a);
ind2_a=ind_a/2;
m2_a=v(ind2_a)
```

- Welchen Datentyp hat *ind2*? Welchen *ind2_a*? Was sind ihre Werte?

C) WAHRHEITSWERTE

Ein weiterer wichtiger Datentyp sind Wahrheitswerte (*logical*).

- Diese können nur zwei Werte annehmen: "wahr", symbolisiert durch *1*, und "falsch", symbolisiert durch *0*.
- Im Matlab-Workspace werden Variablen vom Typ logical durch ein Häkchen symbolisiert.
- Die wahrscheinlich wichtigste Anwendung der Wahrheitswerte ist die oben schon eingeführte Fallunterscheidung. Die Bedingung in der Fallunterscheidung erzeugt einen Wahrheitswert, von dem die weitere Abarbeitung des Programms abhängt.

Logical-Variablen werden definiert

- durch den Befehl *logical*, der eine beliebige Eingabe zu einem Wahrheitswert macht z.B. *a=logical(0)*
- oder durch die oben schon verwendeten Vergleichsoperatoren <, >, ==, <=, >=, ~= , die jeweils einen Wahrheitswert zurückliefern

- oder durch andere Test-Funktionen, die ebenfalls Wahrheitswerte zurückliefern. Besonders wichtig sind dabei die Tests, ob eine Variable einen bestimmten Datentyp hat:
  - *a=isscalar(x) % Wahr für einen skalaren Wert*
  - *a=isvector(x) % Wahr für einen Vektor*
  - *a=isfloat(x) % Wahr für Fließkommazahlen (Vektoren Matrizen, Skalare)*
  - *a=isnumeric(x) % Wahr für alle Zahlen (Vektoren Matrizen, Skalare)*
  - *a=ischar(x) % Wahr für Zeichenketten*
  - *a=islogical(x) % Wahr für Wahrheitswerte*
  - *a=isa(x,'type') % Wahr, wenn x den Datentyp 'type' hat*

Mit Wahrheitswerten kann man "rechnen", indem man logische Verknüpfungen anwendet, um sie z.B. bei den Bedingungen in if-Anweisungen zu kombinieren:

- *c= a&&b % logisches und,*
- *% a und b muessen wahr sein, damit c wahr ist*
- *c= a||b % logisches oder,*
- *% mindestens eine von a und b muss wahr sein, damit c wahr wird.*
- *c= ~a % Negation.*
- *% c ist wahr, wenn a falsch ist.*
- *c= xor(a,b)% Exklusives oder,*
- *% c ist wahr wenn genau eine von a und b wahr ist.*

Aufgaben:

T3C1) Probieren Sie aus:

- *a=logical(0)*
- Welchen Wert und welchen Typ hat a im Workspace?
- Definieren Sie außerdem *b=logical(1)*
- Was passiert bei *c=logical(2)*?

T3C2) Probieren Sie verschiedene Vergleichsoperatoren aus:

- definieren Sie zwei Variablen mit skalaren Zahlenwerten
- vergleichen Sie diese mit jedem der Vergleichsoperatoren<, >, ==, <=, >=, ~=
- (Syntaxtipp: w=(a>b) liefert das gleiche Ergebnis wie w=a>b, ist aber besser lesbar).
- Was ist der Unterschied zwischen = und == ?

- Wenden Sie die Vergleichsoperatoren auch auf zwei Matrizen gleicher Größe an. Welchen Typ hat die Ausgabe? Wie sieht die Rechenregel aus?

T3C3) Es kann sehr hilfreich sein, Vergleichsoperatoren auf Vektoren oder Matrizen anzuwenden, um in Daten diejenigen Messpunkte zu finden, für die bestimmte Bedingungen erfüllt sind. Laden Sie folgende Messreihe des pH Werts von Aquarienwasser: [phWerte.mat].

- Wie viele der Messpunkte sind Messfehler (zumindest wenn Ihre Fische die Messreihe überlebt haben und normale Süßwasserfische sind, die sich normalerweise rund um pH 7 tummeln)?
- Wie häufig wurden zu große und wie häufig zu kleine Werte aufgezeichnet?

T3C4) Manchmal möchte man zwei ganze Matrizen oder Vektoren auf Gleichheit testen und nicht ihre einzelnen Elemente. Dafür gibt es den Befehl *isequal(M,N)*, der einen einzelnen Wahrheitswert zurückliefert.

T3C5) Schreiben Sie eine Funktion, die für einen beliebigen übergebenen Eingabeparameter den Typ der Eingabe zurückgibt.

D) TEXT

Die meisten Programme müssen in irgendeiner Weise mit dem Benutzer interagieren. Die modernste Möglichkeit dafür sind grafische Benutzeroberflächen (GUIs, graphical user interfaces), die man in Matlab relativ komfortabel gestalten kann. Mit diesen werden wir uns im Rahmen des Kurses allerdings nicht beschäftigen. (Aber wenn Sie sich durch die Aufgaben nicht ausgelastet fühlen, können Sie sich gerne nebenbei mit Hilfe eines Tutorials in die Benutzung von GUIs einarbeiten.) Wir beschränken uns in diesem Abschnitt auf die "herkömmliche" Form der Bildschirmausgaben als Text und Abbildungen und der Benutzereingaben per Tastatur im Befehlsfenster.

Wenn man mit Matlab Texte verarbeiten möchte, werden diese als Zeichenketten (array of char) dargestellt.

- Variablen des Typs char werden im Workspace-Fenster durch abc symbolisiert.
- Die Syntax um eine Zeichenkette zu generieren und einer Variablen zuzuweisen ist:
  - *text='Texte werden in Hochkommata eingeschlossen'*
- Mit solchen Zeichenketten kann man in gleicher Weise arbeiten wie mit Vektoren, um einzelne Zeichen oder Zeichenfolgen zu indizieren oder Zeichenketten hintereinander zu hängen.
- Leerzeichen zählen dabei genauso mit wie alle anderen Zeichen.
- Um einen längeren formatierten Text zu erzeugen, kann man auch eine Art Matrix generieren, bei der in jeder Zeile eine Zeichenkette steht.

Allerdings ist dabei zu beachten, dass jede Zeile gleich lang sein muss. Deshalb müssen kürzere Zeilen mit Leerzeichen aufgefüllt werden, um keine Fehlermeldungen zu erhalten. (Das ist manchmal etwas sperrig. Nächste Woche werden wir eine vielleicht etwas elegantere und häufig verwendete Art des Umgangs mit Texten kennen lernen.)

Aufgaben:

T3D1) Probieren Sie aus:

text='Texte werden in Hochkommata eingeschlossen'

text(1)

teil1=text(1:6)

teil2=text(17:25)

syntax=[teil2 ' ' teil1 ' ' teil2]

(damit man den letzten Ausdruck besser lesen kann hier noch mal mit _ für jedes Leerzeichen:

syntax=[teil2_'_'_teil1_'_'_teil2])

T3D2) Wir wissen bereits, dass Berechnungsergebnisse standardmäßig im Command Window ausgegeben werden, wenn sie nicht durch ein *;* am Ende der Zeile unterdrückt werden.

Es kann jedoch sinnvoll sein, dem Benutzer auch andere Informationen per Textausgabe zur Verfügung zu stellen. Dies geschieht durch die Funktion *disp*, der eine Zeichenkette übergeben wird.

- Einfach weil es zu einem Programmierkurs dazugehört, erzeugen Sie ein Skript hello_world.m das als einzige Zeile enthält

disp('hello world')

- Weil das doch ein wenig langweilig ist, ergänzen Sie hello_world so, dass es Ihnen außerdem in freundlicher Art noch das aktuelle Datum verrät. Verwenden Sie dafür die Funktion *date*.

T3D3) Mit einer speziellen Art der Textausgabe sind Sie inzwischen bestens vertraut: den rot geschriebenen Fehlermeldungen. Diese können Sie auch selber erzeugen durch den Befehl error, z.B. mit

error('so geht es ja nun nicht!')

Die Funktion error macht jedoch noch mehr als eine rote Textausgabe, sie bricht außerdem die Ausführung eines Programms ab.

- Probieren Sie das aus, indem Sie ein den error-Befehl an verschiedenen Stellen eines längeren Skripts oder Funktion einfügen (z.B. kompliziert.m von gestern). Wenn Sie in der Datei keine Semikolon verwenden oder den Debugging Modus im Editor benutzen, können Sie nachvollziehen, welche Schritte jeweils ausgeführt werden.

T3D4) Wenn man den Benutzer warnen will, dass gerade wahrscheinlich etwas schief läuft, aber nicht das ganze Programm abgebrochen werden soll, gibt es dafür den Befehl *warning*. Probieren Sie auch diesen aus.

T3D5) Man kann Textausgaben auch mit der Ausgabe von Variablenwerten mischen. Der Befehl dafür ist *sprintf*. Dieses ist eine Funktion, die als erstes Argument eine Zeichenkette übergeben bekommt, die ausgegeben werden soll. Dabei stehen an den Stellen der Variablenwerte die Schlüsselbegriffe *%g* (bei Variablen vom Typ *double, int,* oder *logical*) oder *%s* (bei *string*). Als weitere Argumente werden die Variablennamen in der Reihenfolge übergeben, in der sie im Text auftauchen.

- Probieren Sie aus:

a=0.2;

t='Text';

sprintf('jetzt koennen wir Zahlen wie z.B. %g und auch %s darstellen.',a,t)

- Erweitern Sie eins Ihrer Spaghettiskripte oder Funktionen von gestern so, dass die Ausgabe der Länge mit Angabe der Einheit erfolgt.
- *) Für Leute mit zu viel Zeit oder einem Hang zu schönen Bildschirmausgaben: mit *sprintf* kann man die Formatierung sehr exakt festlegen (z.B. Anzahl Nachkommastellen) - wie das geht steht in der Hilfe. Dort gibt es auch Hinweise zum entsprechenden Befehl *fprintf*, mit dem man formatiert in Dateien schreibt.

T3D6) Zeichenketten werden auch gebraucht, um Abbildungen zu beschriften.

- Plotten Sie für *x1=-10:0.1:10* den Vektor *y1=x1.^2*.
- Benutzen Sie die Befehle *title, xlabel* und *ylabel*, um Ihre Grafik zu beschriften. Diese Befehle bekommen jeweils eine Zeichenkette als Eingabeargument.
- Versehen Sie Ihre Abbildung mit dem Befehl *legend('blablabla')* mit einer Legende (eventuell mit etwas aussagekräftigerem Text).
- Probieren Sie aus, mehrere Kurven in eine Abbildung zu plotten. Hierfür erzeugen Sie für einen zweiten x-Vektor *x2=-5:0.1:5* die y-Werte *y2=2*x2.^2*. (Erinnerung: Um beide Kurven in eine Abbildung zu plotten, muss man jeweils die x- und y-Werte explizit angeben, also in unserem Fall *plot(x1,y1,x2,y2)* oder *hold on* verwenden).
- Beschriften Sie in der Legende beide Kurven. *legend* bekommt als Eingabe so viele durch Komma getrennte Zeichenketten wie Vektoren in der Grafik dargestellt sind.

T3D7) Häufig möchte man dem Benutzer nicht nur etwas mitteilen, sondern auch Benutzereingaben erhalten. Einen Spezialfall davon haben Sie bereits kennengelernt, den Befehl *pause*. Wenn auch der Wert einer Eingabe abgefragt werden soll, verwendet man den Befehl *input*.

a=input('Eingabe') weist der Variablen a eine Zahl oder einen Variablenwert zu. Probieren Sie aus:

a=input('Wert fuer a: ')

-> bei Abfrage *7* eingeben

b=input('Wert fuer b: ')

-> bei Abfrage *a* eingeben

c=input('Wert fuer c: ')

-> bei Abfrage *xyz* eingeben

Wenn Text eingelesen werden soll, braucht der Befehl als zweiten Eingangswert die Option 's':

c=input('Wert fuer c: ','s')

-> bei Abfrage *xyz* eingeben

d=input('Wert fuer d: ','s')

-> bei Abfrage *7* eingeben

e=input('Wert fuer e: ','s')

-> bei Abfrage *b* eingeben

T3D8) Um Zeichenketten zu vergleichen, gibt es den Befehl *strcmp(t1,t2)* (string comparison), der eine logische *1*zurückliefert, wenn die Zeichenketten *t1* und *t2* gleich sind. Alternativ kann man auch den oben schon eingeführten Befehl *isequal* benutzen.

- Probieren Sie *==, isequal* und *strcmp* für gleiche und verschiedene Zeichenketten aus.

T3D9) Man kann Zeichenketten in Zahlen umwandeln und umgekehrt. Die Befehle dafür sind *num2str* (number to string) und *str2num*. Probieren Sie aus:

a=1.1

a_char=num2str(a)

a_neu=str2num(a_char)

b='xyz';

b_num=str2num(b)

- Welche Typen haben die Variablen jeweils?

T3D10) Diese Umwandlung ist besonders praktisch, um Zeichenketten automatisch zu generieren, z.B. für die systematische Benennung von Dateien. Schreiben Sie ein Programm, das

- zwei Eingaben vom Benutzer abfragt:
    1. eine Zeichenkette und
    2. eine Zahl

- Benutzen Sie die eingegebene Zahl, um in Kombination mit der festen Zeichenkette *'meine_datei'* den Dateinamen zu erzeugen. Wenn der Benutzer z.B. *5* eingegeben hat, soll die Datei *'meine_datei5.mat*' heißen.
- Speichern Sie die vom Benutzer eingegebene Zeichenkette in einer Datei diesen Namens.
- Achtung, hier funktioniert nicht die normale Syntax

  save dateiname variablenname
- (Diese würde man erwarten, wenn der generierte Dateiname in der Variablen *dateiname* gespeicher ist). Hierfür braucht man die kompliziertere (und leider sehr wenig intuitive) Syntax

  save(dateiname,'variablenname')

  (Fragen Sie mich bitte nicht, warum hinten um den Variablennamen Anführungsstriche stehen!)

T3D11) Achtung! Matlab hindert Sie nicht daran, mit Zeichenketten zu rechnen! Probieren Sie aus:

```
a='1'
a2=2*a
a2_ok=2*str2num(a)
b='xyz'
b+1
```

T3D12) Um sich Programmtexte im command window anzeigen zu lassen, gibt es den Befehl *type*. Probieren Sie in Ihrem Arbeitsverzeichnis aus:

- *type spaghettiskript*
- Praktischerweise gilt *type* nicht nur für Ihre eigenen Programme, sondern auch für die in Matlab enthaltenen, so dass man sich diese ohne lange Suche ansehen und sich etwas abgucken kann. Probieren Sie z.B. *type imagesc* (nein, den Programmcode müssen Sie noch nicht verstehen).
- Bei manchen Kern-Funktionen lässt Matlab sich allerdings nicht auf die Finger schauen, z.B. *type sin*

H) HAUSAUFGABEN

*T3H1) Probieren Sie aus, welche Datentypen sich in welche umwandeln lassen.

- Erzeugen Sie je eine Variable der verschiedenen Ihnen bekannten Datentypen und wandeln sie jeweils den Typ in die anderen Datentypen.
- Was passiert z.B. wenn man aus Zeichen Zahlen macht und umgekehrt?
- Was passiert mit Vorzeichen und Nachkommastellen, wenn man Umwandlungen in andere Typen vornimmt?

T3H2) In folgender Datei sind die Merkmale der Katzen in einem Tierheim gespeichert: [katzen.mat].

- Im Vektor *alter* ist für jede Katze ihr Alter in Jahren gespeichert,
- im Vektor *weibchen* ein *true*, wenn es sich bei der Katze um ein Weibchen handelt, bei Katern ein *false*.
- Die Merkmale der Tiere sind jeweils in der gleichen Reihenfolge angeordnet.
- Wie bekommt man heraus, wie viele Katzen jünger sind als 2 Jahre?
- Wie die Anzahl der unter zweijährigen Weibchen?

T3H3) ABGABE! Bei den Merkmale der Katzen im Tierheim [katzen.mat] findet sich auch eine Matrix *farben*. Diese besteht aus Wahrheitswerten und ist folgendermaßen aufgebaut:

Zeilen: 23 Katzen

Spalte 1: Katze hat schwarzes Fell

Spalte 2: Katze hat rotes Fell

Spalte 3: Katze hat weißes Fell

Spalte 4: Katze hat getigertes Fell

- Wie viele Katzen haben getigertes Fell?
- Wie viele Katzen haben rein getigertes Fell, ohne andere Farben?
- Wie viele dreifarbige Glückskatzen (schwarz-weiß-rot) gibt es?
- Wie viele einfarbige Katzen gibt es (wenn getigerte Katzen nicht als einfarbig angesehen werden)?

T3H4) ABGABE! 6 Punkte Schreiben Sie ein Programm, das Besuchern des Tierheims durch einen Dialog ermöglicht herauszufinden, ob es ihre Traumkatze bezüglich Alter, Geschlecht und Fellfarbe im Tierheim gibt.

(** Bonuspunkte möglich: Die letzte Aufgabe ist beliebig ausbaubar. Sie können sich selber überlegen, wie viele Details Sie in Ihrem Programm bedenken wollen.)

Achten Sie dabei auf

- Benutzerfreundlichkeit: Die Benutzer müssen wissen, was sie zu tun haben, um vernünftig mit dem Programm zu interagieren
- Robustheit: Das Programm sollte auf „falsche" Eingaben des Benutzers vernünftig reagieren
- Korrektheit: Es versteht sich wahrscheinlich von selbst, dass die Angaben, die das Programm über die Datenbasis macht, korrekt sein sollen.

*T3H5) Für Leute mit zu viel Zeit oder gesteigertem Interesse an Programmierkonzepten:

Matlab sieht für Abfragen von vielen verschiedenen Fällen eine switch-Anweisung vor. Schauen Sie in der Hilfe nach, wie diese zu verwenden ist

(eine Kurzversion finden Sie auch in der Befehlsreferenz), und schreiben Sie eine Funktion "arbeitsamt", die einen Namen (*string*) als Eingabe bekommt und diesen Namen dem zuständigen Sachbearbeiter zuweist und dessen Namen auf dem Bildschirm ausgibt. Zuständigkeit für Anfangsbuchstaben:

A-D: Frau Schmidt
E-H: Herr Klein
I-L: Frau Gross
M-P: Frau Mueller
Q-T: Herr Maier
U-Z: Herr Dimpfelmoser

*T3H6) Zur Vertiefung: Schauen Sie sich dieses Programm aus einem anderen Kurs an, in dem z.B. auch geschachtelte Abfragen eingeführt werden: if_demo.m