Kapitel 10

Funktionen

Funktionen

- Eine Funktion ist die Zusammenfassung von mehreren Anweisungen unter einem bestimmten Namen.
- Funktionen verwendet man, um mehrfach benötigte Anweisungsfolgen nur einmal niederzuschreiben und von unterschiedlichen Stellen aufzurufen.
- Mit Funktionen kann ein großes Programm in viele kleine Teile (Unterprogramme) aufgeteilt werden.

Funktionsdefinition

- Eine Funktion besteht aus einem Namen, eventuellen Übergabeparametern (oder auch Eingabeparametern) und eventuellen Rückgabeparametern (Ausgabeparametern).
- Alle Variablen, die nur in der Funktion benutzt werden, sind nach Verlassen der Funktion nicht mehr vorhanden.
- Vor Matlab Version R2016b musste jede Funktion in ihre eigene m-Datei geschrieben werden, die auch genauso heißen musste, wie die Funktion.
- Seit Matlab Version R2016b können Funktionen auch an das Ende der m-Datei gestellt werden. Somit sind auch mehrere Funktionen in der m-Datei möglich.

Funktionsdefinition

- Formaler Aufbau einer Funktion:

```
function [out1,out2,...,outn] = Funktionsname(arg1, arg2, ..., argn)
    % Beschreibung der Funktion als Kommentar
    Anweisungen
end
```

Funktionsdefinition

- Formaler Aufbau einer Funktion beginnt mit dem Schlüsselwort "function". Danach sind durch Komma getrennt und in [], die Rückgabeparameter aufgelistet.
- Auf der rechten Seite des = steht der Funktionsname mit in runden Klammern aufgeführten Übergabeparametern.
- In der Funktion kann zuerst ein Kommentar stehen. Dieser wird angezeigt, wenn man "help" zu dieser Funktion aufruft.
- Dann folgen die Anweisungen, die die Funktion ausführen soll.
- Die Funktion wird durch das Schlüsselwort "end" abgeschlossen.

Eine Funktion benutzen

- Eine Funktion wird aufgerufen (benutzt), indem man den Namen der Funktion angibt.
- Hat die Funktion Übergabe- und/oder Rückgabeparameter, so muss man diese Variablen entsprechend auch benennen/angeben.

Funktion mit Parametern - Beispiel 1

```
quadrieren(3)


function xquad = quadrieren(x)
        xquad = x^2;
end
```

```
ans =
           9
```

Funktion mit Parametern - Beispiel 1

- Die gezeigte Funktion besitzt einen Übergabeparameter "x" und einen Rückgabeparameter "xquad".
- Durch den Aufruf

  quadrieren(3)

  wird zur Funktion gesprungen und die 3 an die Variable x übergeben.
- Nun kann die Rechnung xquad = x^2 ausgeführt werden. Das Ergebnis wird hierbei in der Variablen xquad gespeichert und bei Beendigung zurück und daraufhin im Command Window ausgegeben

Funktion mit Parametern - Beispiel 2

```
[volumenErg, umfangErg] = kugelBerechnungen(3);
disp(volumenErg);
disp(umfangErg);

function [volumen, umfang] = kugelBerechnungen(radius)
    volumen = 4.0 / 3.0 * 3.1415 * radius^3;
    umfang = 2 * 3.1415 * radius;
end
```

```
113.0940
18.8490
```

Funktion mit Parametern - Beispiel 2

- Diese Funktion besitzt einen Übergabeparameter "radius" und zwei Rückgabeparameter "volumen" und "umfang".
- Man sieht, dass die Rückgabeparameter, (sollten es mehr, als einer sein,) in [] stehen müssen.
- Um das Ergebnis (die Werte der zwei Variablen volumen und umfang) aufnehmen zu können, sind beim Aufruf der Funktion ebenso zwei in [] stehende Variablen anzugeben.
-

```
[volumenErg, umfangErg] = kugelBerechnungen(3);
```

Funktion mit Parametern - Beispiel 3

- In folgendem Beispiel wird die Berechnung des Volumens in eine eigene Funktion ausgelagert.
- Das Hauptprogramm kümmert sich somit nur um die Ein- und Ausgabe.
- Siehe Beispiel 11 (auf der Webseite zu finden).

Funktion ohne Parameter - Beispiel 1

- Eine Funktion kann, aber muss keine
  - Übergabeparameter

  und/oder auch keine
  - Rückgabeparameter

  haben.

```
ausgeben()

function ausgeben()
    disp("Hello World");
end
```

```
Hello World
```

Der Befehl *return* bei Funktionen

- Eine Funktion kann durch den Befehl *return* früher verlassen werden. "Berechnung beendet" würde dann nicht mehr angezeigt werden.

```
x = input("x eingeben: ");
xquad = quadrieren(x)

function xquad = quadrieren(x)
    if (x < 10)
       xquad = x^2;
    else
        xquad = 'ohne Wert';
        return;
    end
    disp("Berechnung beendet");
end
```

Funktionsparameter

- Matlab arbeitet nach dem Prinzip "lazy copying".
- Werden einer Funktion Werte übergeben, wäre es ineffizient und speicher-verschwendend, wenn alle Variablen als Kopie gehalten werden würden.
- Daher analysiert Matlab, welche übergebenen Variablen in der Funktion verändert werden. Nur diese Variablen werden dann als Kopie übergeben; existieren also mehrmals im Speicher.
- Variablen, die in der aufgerufenen Funktion nicht geändert werden, werden nur als Verweis (Referenz) übergeben; existieren also nur einmal im Speicher.

Funktionsparameter

```
function y = test(x, a, b)
    a = a + 12;
    y = a * x + b;
end
```

- Hier wird nur die Variable a verändert.
- Die Variablen b und x dagegen nicht.
- Das bedeutet, dass a im Hauptprogramm und als Kopie in der Funktion existiert.
- b und x dagegen existieren nur einmal im gesamten Programm

Eine Konstante als Funktion

- In Matlab kann man keine richtigen Konstanten deklarieren.
- Die Variablen, die man deklariert, sind immer veränderbar.
- Um in Matlab aber trotzdem eine unveränderbare Variable zu erschaffen und ein versehentliches Ändern des Wertes zu verhindern, packt man diese Variable in eine eigene Funktion.

```
function PI = constPI()
    PI = 3.14159;
end
```

- Dies kann dann so benutzt werden:

```
volumen = 4 / 3 * constPI() * radius^3;
```

Funktion mit beliebig vielen Parametern

- Es gibt in jeder aufgerufenen Funktion die allgemeingültigen Variablen:
  - varargin = Variable-length input argument list
    (Die Übergabeparameter in beliebiger Anzahl)
  - varargout = Variable-length output argument list
    (Die Rückgabeparameter in beliebiger Anzahl)
  - nargin = Number of function input arguments
    (Die Anzahl der Übergabeparameter)
  - nargout = Number of function output arguments
    (Die Anzahl der Rückgabeparameter)

Funktion mit beliebig vielen Parametern

- Man kann eine Funktion auch so definieren, dass sie beliebig viele Übergabe- und auch Rückgabe-Parameter haben kann.
- Dabei sind die allgemeingültigen Variablen von großem Nutzen.

- In folgendem Beispiel wird eine Funktion mehrmals mit unterschiedlicher Parameteranzahl aufgerufen.
- Siehe Beispiel 12 (auf der Webseite zu finden).

Funktionen - verschiedene Typen

- Es gibt 4 verschiedene Funktionstypen:
  - *Anonymous Functions*
  - *Local Functions*
  - *Nested Functions*
  - *Private Functions*

(Anonymous und Private Functions werden wir hier nicht weiter behandeln. Für weitere Informationen siehe Matlab-Hilfe.)

main, local (sub) functions

- In eine m-Datei kann man eine oder mehrere Funktionen schreiben.
- Hierbei wird die erste Funktion *main function* genannt.
- Diese *main function* muss am Anfang der Datei stehen und genauso heißen, wie die Datei.
- Sie kann dann von anderen Dateien oder vom "Command Window" aus aufgerufen werden.
- Gibt es in dieser m-Datei weitere Funktionen, so werden diese *local function* genannt. Sie können nur von Funktionen in dieser Datei aufgerufen werden. Daher nennt man sie manchmal auch *sub function*.

main, local (sub) functions

quadrieren.m:

```
function xquad = quadrieren(x)
    if (x < 10)
       xquad = x^2;
    else
        xquad = noValue();
        return;
    end
end

function result = noValue()
    result = '-kein Wert-';
end
```

nested functions

- Steht eine Funktion in einer anderen Funktion, so nennt man sie *nested function*.
- Diese nested functions haben Zugriff auf die Variablen der übergeordneten Funktion; auch, wenn diese nicht als Parameter übergeben werden.

nested functions

```
main

function main
    x = 5;
    nestedfunc;

    function nestedfunc
        x = x + 1;
    end
    disp(x);
end
```

```
6
```