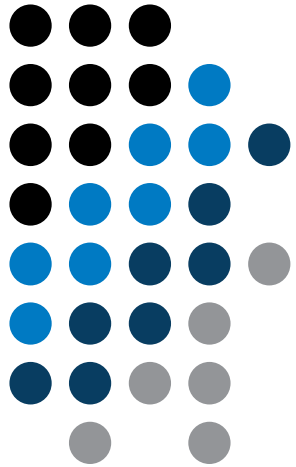
# AE0B17MTB – Matlab

# Part #8

Miloslav Čapek
miloslav.capek@fel.cvut.cz
Filip Kozák, Viktor Adler, Pavel Valtr

Department of Electromagnetic Field
B2-626, Prague

# Learning how to …

**Strings**

**`eval`, `feval`**

**Matlab `path`**

# Strings in Matlab

- string = array (a vector or a matrix or a cell) of characters
  - Try to avoid diacritics (accent) in Matlab
- string is created using apostrophes

```
>> st = 'Hello, world!'
```

  - strings are outputs of some functions (e.g. `>> char(65)`)
- each character in a string is an element of an array and requires 2B
  - datatype `char`
- when an apostrophe is required to be part of a string, it is to be typed as two quote characters:

```
>> pt = 'That''s it!'
```

# Strings – principles

- in the case string has more than one line, it has to have same number of columns

```
>> st = ['karel'; 'pepi ']
```

- otherwise (usually) strings are stored as cell datatype:

```
pt = {'karl', 'pepi', 'and all others', 'including accents ěščř'}
```

- whether a given variable is of type char is tested this way:

```
>> ischar(st)
>> iscellstr(pt)
```

# Strings - type conversion

- quite often, it is required to convert from a number code to a string and vice versa, e.g.
  - `double` → `char`
  - `char` → `double`
  - `char` → `uint16`

```
>> tx = char([65:70])

>> B = double(tx)

>> C = uint16(tx)

>> whos
```

- operations with strings are similar to operations with numerical arrays
  - holds true for indexing in the first place!

```
>> S1 = 'test'; S2 = '_b5';
>> S3 = [S1 S2]
>> size(S3), size(S3')
>> S4 = [S3(3:5) 'end']
```

# Strings

200 s ↑

- create an arbitrary string
  - find out its length
  - try to convert the string into double type
  - try to index selected parts of the string

- questions???

# Strings – number conversion #1

- conversion of number in a string (`char`) to number (`double`):
  - conversion of multiple numbers (function `str2num`):

```
>> str2num('[1 2 3 pi]')
>> str2num('[1, 2;3 4]')
```

```
>> str2num('[1 2 3 pi]')

ans =

    1.0000    2.0000    3.0000    3.1416
```

```
>> str2num('[1, 2;3 4]')

ans =

     1     2
     3     4
```

  - conversion of a single number to `double` (`str2double`):

```
>> str2double('1 +1j')
>> str2double('-0.5453')
```

  - pay attention to possible errors that should be treated in the code

```
>> str2num('1a')
ans =
     []
```

```
>> str2num('1+1j')
```

```
>> str2num('1 +1j')
```

```
>> str2double('[1 2 3 pi]')
ans =
    NaN
```

```
>> str2num('1+1j')

ans =

   1.0000 + 1.0000i
```

```
>> str2num('1 +1j')

ans =

   1.0000 + 0.0000i   0.0000 + 1.0000i
```

# Strings – number conversion #2

- quite often it is needed to convert numerical result back to a string

```
>> num2str(pi)
>> num2str(pi, 10)
```

```
>> disp(['the value of pi is: ' num2str(pi, 5)]);
```

- for listing purposes it is advantageous to use the function `sprintf`
  - it enables to control output format in a better way

```
>> st = sprintf('the value of pi is: %0.5f\n', pi);
>> st
```

# Strings – other conversions

- among others there are other functions available

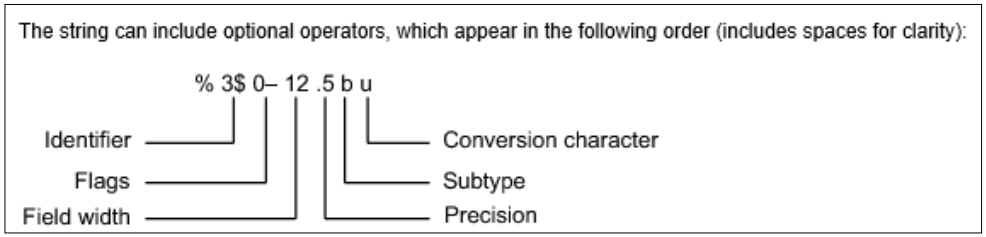
| Function | Description |
|---|---|
| `int2str` | convert integer to text; in the case the input parameter is not an integer its value it is rounded first |
| `mat2str` | converts matrix to string |
| `hex2num, num2hex` | converts hexadecimal number of type `char` to a number (and vice versa) |

- e.g.

```
>> mat2str(magic(3))
```

# Strings – formatting

- function `sprintf` generates a string with given formatting
  - for more see `>> doc sprintf`
  - alternatively, `disp(sprintf(...))`




- function `fprintf` writes string
  - on a screen (`fid` = 1 / 2)
  - in a file (`fid` to be obtained e.g. using function `fopen`, see later)

```
>> st = sprintf('the value of pi je: %2.3e\n\n', pi);
>> fprintf(st) % or directly fprintf('...', pi);
```

```
>> fprintf(fid, st)
```

# Strings

450 s ↑

- create following strings using `sprintf` help:
  - I.

```
ans =

Hodnota pi je: 3.14159, hodnota 5*pi je: 15.70796
```

  - i.e. both numbers are displayed with five digits accuracy
  - II.

```
ans =

Toto je 50%
```

  - i.e. display percent sign, the expression contains 3 line spacings
  - III.

```
ans =

Toto je sada měření: test_A
```

  - i.e. insert variable into the string, value of which is `'test_A'` string

# Strings

200 s ↑

- think about the differences between `disp` a `fprintf` (`sprintf`)
  - describe the differences
  - what function do you use in a particular situation?

# Type conversion (general) – a note

- Matlab determines data types by itself
  - and also performs type conversion if needed

- single / double precission: `single()` / `double()`

- if, however, a particular data type is required that was not assigned on creation of a variable, this variable can by type-converted :
  - function `cast`: performs type conversion, values are truncated as the case may be
  - function `typecast`: performs type conversion and keeps the size of the original variable from the memory point of view as well as the bit value
  - see Matlab documentation for more

# Upper case / lower case characters

- `lower` converts all letters in strings to lower case

```
>> lower('All will bE LOWERCASE')
% ans =
% all will be lowercase
```

- `upper` converts all letters in strings to upper case

```
>> str = 'all will be upper case';
>> str = upper(str)
% str =
% ALL WILL BE UPPER CASE
```

  - support of characters from Latin 1 character set on PCs
  - other platforms: ISO Latin-1 (ISO 8859-1)
  - $\Rightarrow$ supports Czech accents

# Strings – searching

- `strfind` finds a given string inside another
  - returns indexes (positions)
  - searches for multiple occurrences
  - is CaSe sEnSiTiVe
  - enables to search for spaces etc.

```
>> lookFor = 'o';
>> res = strfind('this book', lookFor);
res =
     7     8
```

# Strings – comparing

- two strings can be compared using function `strcmp`
  - the function is often used inside `if-else` / `switch-case` statements
  - the result is either `true` or `false`
  - it is possible to compare string vs. `cell` of strings or `cell` vs. `cell`

```
>> strcmp('tel', 'A')
>> strcmp('tel', 'tel')
>> strcmp('test', {'test', 'A', '3', 6, 'test'})
>> strcmp({'A', 'B'; 'C', 'D'}, {'A', 'F'; 'C', 'C'})
```

$$\left( \begin{array}{|c|c|}\hline \mathbf{A} & \mathbf{B} \\ \hline \mathbf{C} & \mathbf{D} \\ \hline \end{array} \;\; == \;\; \begin{array}{|c|c|}\hline \mathbf{A} & \mathbf{F} \\ \hline \mathbf{C} & \mathbf{C} \\ \hline \end{array} \right) = \begin{array}{|c|c|}\hline \mathbf{1} & \mathbf{0} \\ \hline \mathbf{1} & \mathbf{0} \\ \hline \end{array}$$

# Strings – joining

- strings can be joint together using function `strjoin`
  - it as applicable to variables of type `cell`
  - separator is optional (implicitly a space character)

```
>> cl = {'A', 'B', 'C', 'D'}
>> strjoin(cl)
>> strjoin(cl, ',')
```

```
>> cl = {'A','B','C','D'}

cl =

    'A'    'B'    'C'    'D'

>> strjoin(cl)

ans =

A B C D

>> strjoin(cl,',')

ans =

A,B,C,D
```

- `fullfile` connects individual folders into a file path
  - back slash (\) is inserted between individual items

```
>> folder1 = 'Matlab';
>> folder2 = 'project_one';
>> file    = 'run_process.m';
>> fpath = fullfile(folder1, folder2, file);
```

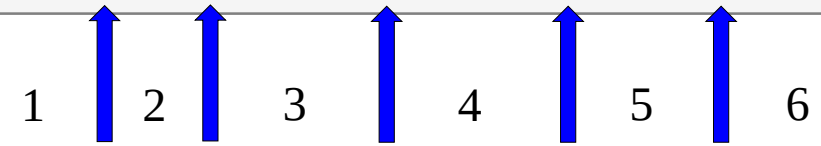
```
fpath =

Matlab\project_one\run_process.m
```

  - will be used for exporting and work with GUI

# Strings – string separation

- function `deblank` removes excess space characters from end of string
- function `strtrim` removes space characters from beginning and end of string
- if a string is to be split, function `strtok` is used
  - separator can be chosen arbitrarily

```
>> this_str = 'some few little little small words'
```




```
>> [token, remain] = strtok(this_str, ' ');
```

first separated word

rest of string

# Strings – string separation

- function `regexp` enables to search a string using regular expressions
  - syntax of the function is a bit complicated but its capabilities are vast!!
    - Ex.: search for all words beginning with '`wh`' with vowels `a` or `e` after and containing 2 characters:

```
>> that_str = 'what which where whose';
>> regexp(that_str, 'wh[ae]..', 'match')
```

    - Ex.: search indexes (positions) where words containing `a` or `o` begin and end

```
>> that_str   = 'what which where whose';
>> [from, to] = regexp(that_str, '\w*[ao]\w*')
```

  - for more details see `>> doc regexp` → `Input Arguments`
  - in combinantion with above mentioned functions, typical tokenizer can be created

# Strings

600 s ↑

- try out following commands and try in advance to estimate what happens …

```
>> str2num('4.126e7')
>> str2num('4.126A')
>> D = '[5 7 9]';
>> str2num(D)
>> str2double(D)
>> int2str(pi + 5.7)
>> A = magic(3);
>> mat2str(A)
>> disp([15 pi 20-5i]);
>> disp(D);
>> B = 'MaTLaB';
>> lower(B)
```

```
>> C = 'cik cak cet  ';
>> findstr(C, 'cak')
>> deblank(C)
>> [tok remain] = strtok(C, ' ')
>> [st se] = regexp(C, 'c[aeiou]k')
>> [st se] = regexp(C, 'c[ei][kt]')
>> regexp(C, '[d-k]')
>> fprintf('Result is %3.7f', pi);
>> fprintf(1, 'Enter\n\n');
```

```
>> disp([' Result: ' num2str(A(2, 3)) 'mm']);
>> fprintf(1, '% 6.3f%% (per cent)\n', 19.21568);
>> fprintf('Will be: %3.7fV\n', 1e4*(1:3)*pi);
>> fprintf('A=%3.0f, B=%2.0f, C=%1.1f\n', magic(3));
>> fprintf('%3.3e + %3.3f = %3.3f\n', 5.13, 13, 5+13);
>> fprintf(1, '%s a %s\n\n', B, C([1:3 5:7]));
```

# Strings – comparing

300 s ↑

- function to compare strings (CaSe SeNsItIvE) is called `strcmp`
  - try to find a similar function that is case insensitive

```
>> strcmpi(string1, string2)
```

  - try to find a function that is analogical to the above one (i.e. case insensitive), but compares first `n` characters only

```
>> strncmpi(string1, string2)
```

  - think about alternatives to the `strcmp` function

```
>> isequal(string1, string2)
>> all(string1 == string2)
```

# Strings

300 s ↑

- remove all blank spaces from the following string
  - try to recollect logical indexing
  - or use an arbitrary Matlab function

```
>> s = 'this is a big book'
```

# Strings

420 s ↑

- write a script/function that splits following sentence into individual words
  - modify the code so that number of occurrence of the string `'is'` was displayed
  - list the words individually including position of the word within the sentence (use `fprintf`)

```
This-sentence-is-for-testing-purposes-only.
```

# Strings


- write a script/function that splits following sentence into individual words
- the problem can be solved in a more elegant way using function `textscan`
  - solution, however, is not complete (word order is missing)

# `eval` – string as a command

- motivation:

```
>> st  = 'sqrt(abs(sin(x).*cos(y)))';
>> x   = 0:0.01:2*pi;
>> y   = -x;
>> fxy = eval(st);
>> plot(x, fxy);
```

  i.e. there is a string containing executable terms
  - its execution is carried out by function `eval`
- applicable mainly when working with GUI (execution of commands entered by user, processing callback functions etc.)
- `eval` has certain disadvantages, therefore its usage is a matter of consideration:
  - block of code with `eval` is not compiled (slow down)
  - text inside the string can overwrite anything
  - syntax inside the string is not checked, it is more diffuclult to understand
- see function help for casees where it is possible to replace `eval`
  - e.g. storing files with serial number (`data1.mat`, `data2.mat`, …)

# evalc

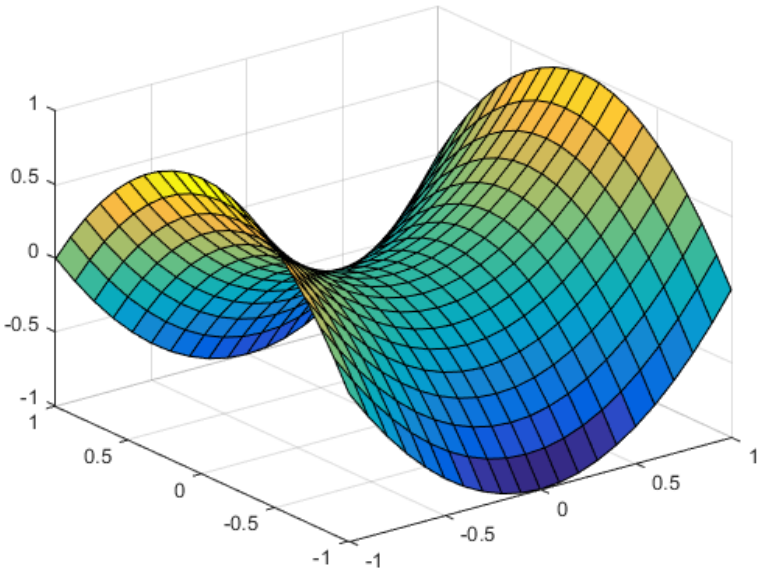
- in some cases it is needed not only to carry out a command in form of a string but also to store the result of the command for later use
- funkce `evalc` („*eval with capture*“) serves this purpose

```
>> CMD = evalc(['var = ' num2str(pi)]);
>> CMD

CMD =


var =

    3.1416



>> whos
  Name       Size            Bytes  Class     Attributes

  CMD        1x20               40  char
  var        1x1                 8  double
```

# `feval` – evaluation of a handle function

- the function is used to evaluate handle functions
  - simply speaking, where `eval` evaluates a string there `feval` evaluates function represented by its handle
  - consider this task:

$$f(x,y) = x^2 + y^2, \quad x, y \in \langle -1, 1 \rangle$$




```
>> hFcn    = @(x,y) x.^2 - y.^2;
>> x       = -1:0.1:1;
>> y       = x;
>> [X, Y] = meshgrid(x, y);
```

```
>> fxy     = hFcn(X, Y);
>> surf(X, Y, fxy);
```

```
>> fxy     = feval(hFcn, X, Y);
>> surf(X, Y, fxy);
```

# Newton's method – modification

600 s ↑

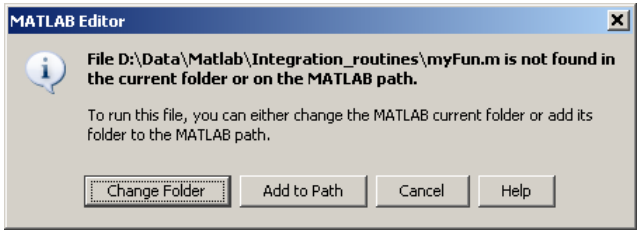
- modify former Newton's method applied to finding root of a polynomial in the way that the polynomial is entered in the form of a handle function
  - see `elmag.org/cs/Matlab/harmonogram` → week 7
  - verify the code by finding roots of following polynomials :

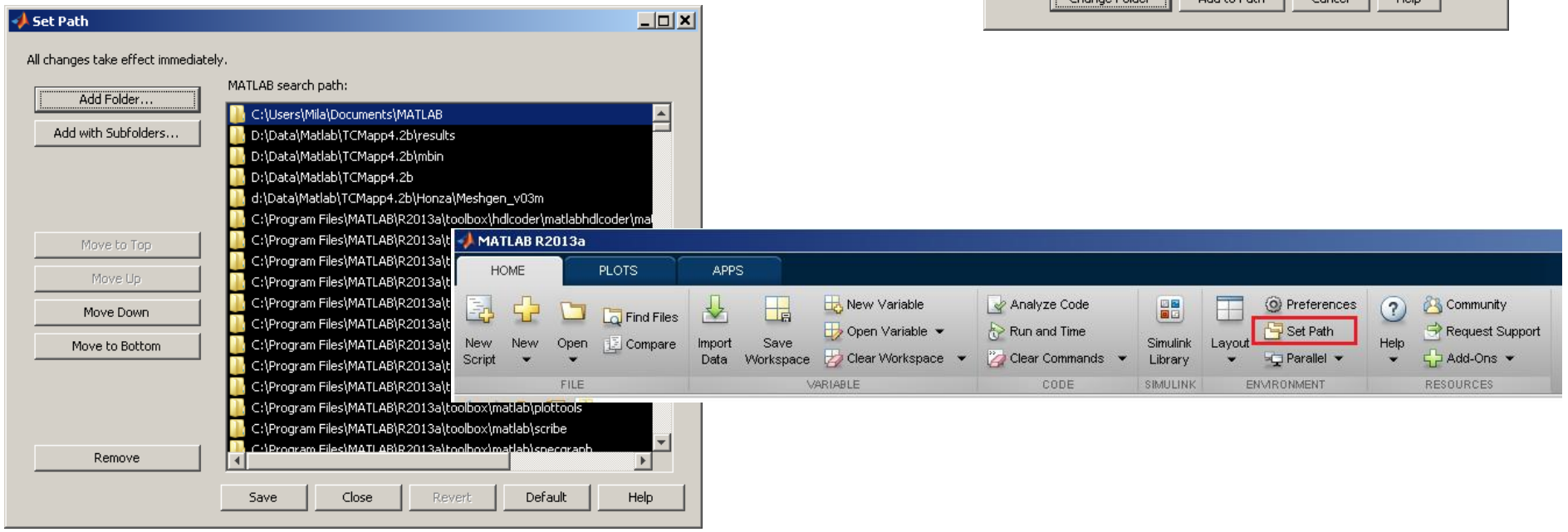
$$x - 2 = 0, \quad x^2 = 1$$

  - verify the result using function `roots`

# Matlab `path`

- list of directories seen by Matlab : `>> path`
- for more see `>> doc path`
- `addpath`: adds folder to path
- `rmpath`: removes folder from path



# Calling a function – order

- how Matlab searches for a function:
  - it is a variable
  - function imported using `import`
  - nested or secondary function inside given function
  - private function
  - function (method) of a given class or constructor of the class
  - function in given folder
  - function anywhere within reach of Matlab (`path`)
- Inside a given folder is the priority of various suffixes as follows:
  - built-in functions
  - `mex` functions
  - Simulink functions (`slx` / `mdl`)
  - `p`-files
  - `m`-files

# Function `exist`

- the function finds out whether the given word corresponds to existing

  (=1) variable in Matlab Workspace

  (=5) built-in function

  (=7) directory

  (=3) mex/dll function/library

  (=4) mdl-file

  (=6) p-file

  (=2) m-file known to Matlab (including user functions, if visible to Matlab)

  (=8) class

  (in the order of priority, return value in bracket)

```
>> type = exist('sin')    % type = 5
>> exist('task1', 'var')  % is the file task1 ...
>> exist('ukol1', 'dir')  % a variable / ...
>> exist('ukol1', 'file') % directory / file?
```

# What does your m-file depend on?

- in the case you compile your code, send it to colleagues etc., it is suitable to test whether they have all files and functions required

- use function `depfun`
  - Ex.1: function sinus `(sin)`

```
>> depfun('sin')
============================================================
depfun report summary:
------------------------------------------------------------
-> trace list:           1 files  (total)
                         1 files  (total arguments)
                         0 files  (arguments off MATLABPATH)
                         0 files  (argument duplicates on MATLABPATH)
------------------------------------------------------------
Notes: 1. Use argument  '-quiet' to not print this summary.
       2. Use arguments '-print','file' to produce a full
          report in file.
       3. Use argument  '-all' to display all possible
          left hand side arguments in the report(s).
============================================================

ans =

    {''}
```

- analogously there is the function `depdir`

# depfun

- starting with Matlab R2014b function `depfun` is replaced by
  - `matlab.codetools.requiredFilesAndProducts`

- Ex.2: Newton's method

```
>> depfun('newton_method')
============================================================
depfun report summary:
------------------------------------------------------------
-> trace list:          30 files  (total)
                         1 files  (total arguments)
                         0 files  (arguments off MATLABPATH)
                         0 files  (argument duplicates on MATLABPATH)
------------------------------------------------------------
Notes: 1. Use argument  '-quiet' to not print this summary.
       2. Use arguments '-print','file' to produce a full
          report in file.
       3. Use argument  '-all' to display all possible
          left hand side arguments in the report(s).
============================================================
```

```
ans =

'd:\Data\Matlab\newton_method.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\@opaque\char.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\@opaque\double.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\@opaque\toChar.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\num2cell.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\elmat\fliplr.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\elmat\repmat.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\@opaque\disp.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\@opaque\display.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\@opaque\evalc.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\ans.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\details.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\ops\@opaque\eq.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\ops\@opaque\ne.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@cell\strcat.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@cell\strfind.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@cell\strjust.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\findstr.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\isspace.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\private\fromOpaq
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\strcmp.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\strfind.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\strncmp.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\blanks.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\cellstr.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\int2str.m'
```

# How to create a function – tips

- how to indicate that the given function / script is running?
  - try several possibilities below…

```
fprintf('START\n    ');
for n = 1:100
    fprintf(1, '\b\b\b\b%3.0f%%', n);
    pause(0.05);
end
fprintf('\nEND\n');
```

```
T = ['/' '-' '\'];
fprintf(2, 'START\n\n');
for n = 1:100
    fprintf(1, '\b%c', T(mod(n, 3)+1));
    pause(0.05);
end
fprintf('\b');
fprintf(2, 'END\n');
```

```
fprintf(2, 'START\n');
for n = 1:100
    fprintf(1, '*');
    pause(0.05);
end
fprintf(1, '\n');
fprintf(2, 'END\n');
```

  - later we will see graphical options as well!

# Matlab – file suffix

| suffix | description |
|---|---|
| `.fig` | Matlab figure |
| `.m` | script / function / class |
| `.mat` | binary data file |
| `.mdl, .slx` | Simulink model |
| `.mdlp, .slxp` | Simulink protected model |
| `.mexa64, .mexmaci64, .mexw32, .mexw64` | mex libraries |
| `.mlappinstall` | APP soubor – installer |
| `.mlpkginstall` | support package – installer |
| `.mltbx` | toolbox file – installer |
| `.mn` | MuPAD notebook |
| `.mu` | MuPAD code |
| `.p` | protected Matlab code |

# Discussed functions

| | | |
|---|---|---|
| `char, uint16, ...` | type conversion / creation of variables of given type | ● |
| `single, double` | single / double precision | |
| `ischar, iscellstr` | test if input is character array / cell array of strings | |
| `int2str, mat2str, hex2num, num2hex` | conversion (integers – strings, hexadecimal – IEEE double) | |
| `str2double` | string to double | |
| `sprintf, fprintf` | String formatting, write to text file | ● |
| `cast, typecast` | type conversion (not keeping / keeping underlying size) | |
| `lower, upper` | convert string to lowercase / uppercase | |
| `strfind, strcmp, strjoin, fullfile` | search, compare, join strings | ● |
| `deblank, strtrim, strtok` | remove blank spaces, remove leading and trailing space, split string | ● |
| `regexp, textscan` | search string (including regular expressions) | ● |
| `eval, feval` | evaluate string / evaluate handle function | ● |
| `path, exist, depfun` | view or change search path, check existence of variable | |

# Exercise #1, #2

450 s ↑

- find out how many spaces there are in the phrase „*how are you?*“
  - look in this lecture / help and find out a suitable function

- convert following string to lowercase and find number of characters

```
>> st = 'MATLAB is CaSe sEnSiTiVe!!!';
```

# Exercise #3

300 s ↑

- create a function to calculate volume, surface area and space diagonal of following bodies: cuboid, cylinder
  - the main function `body.m` contains verification of input variables (type, size) and checking whether user wants to calculate cuboid (parameters `'cuboid'`,a,b,c) or cylinder (`'cylinder'`,r,h)
- sub-functions `cuboid()` and `cylinder1()` calculate required parameters

```
function [V, S, u] = body(bod, a, b, c)
% decision making
% call functions
end

function [V, S, u] = cuboid(a, b, c)
% ... code
end

function [V, S, u] = cylinder1(r, h)
% ... code
end
```

# Exercise #3

# Exercise #4

600 s ↑

- create so called tokenizer (text analyzer), that
  - reads a text input `str` entered by user using function `input`
  - reads separator `sep` (attention, space requires some care!!)
  - split `str` in individual parts depending on `sep`
  - store individual parts separately in a variable of type cell
  - analyze how many vowels (a/e/i/o/u) each individual word contains, store this number and display it together with list of all individual words
  - all commands in the whole script / function have to be terminated with a semicolon!

# Exercise #4

- create a tokenizer (text analyzer)
  - solution using `strtok`

# Exercise #4

- improved solution using `strsplit`

# Exercise #5

600 s ↑

- try to create simple unit convertor, length `x` is given in `'mm'`, `'cm'`, `'in'`, `'inch'` (variable `units`), length in inches can be marked as `'in'` or `'inch'`. Length will be transformed into [mm] according to entered unit string.
  - what decision making construct are you going to use?
  - add a statement from which unit the length was converted and what the result is

```
x     = 15;
units = 'in';
% add the rest
```

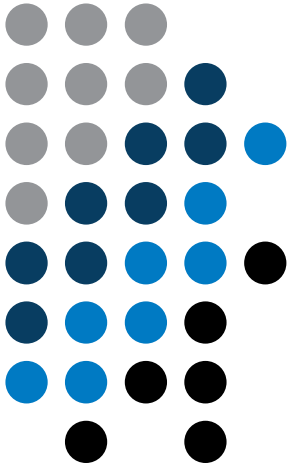
# Exercise #5

# Unit conversion – more elegant way

- use data type `struct` and its properties
  - individual arrays in the structure can be indexed using variables of type `char`

```
function result = convertLength(in_val, in_unit, out_unit)

% supported units for conversion
conversion.in   = 1e4/254; % en.wikipedia.org/wiki/Imperial_units
conversion.inch = conversion.in;
conversion.mm   = 1e3;
conversion.cm   = 1e2;
conversion.m    = 1;

% are the units supported?
if ~isfield(conversion, in_unit)
    error('convertor:nonExistentUnit', ['Unknown unit: ' in_unit]);
end

% calculation
result = in_val * conversion.(out_unit) / conversion.(in_unit);
```

# Thank you!

ver. 4.2 (28/09/2015)
Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz
Pavel.Valtr@fel.cvut.cz