

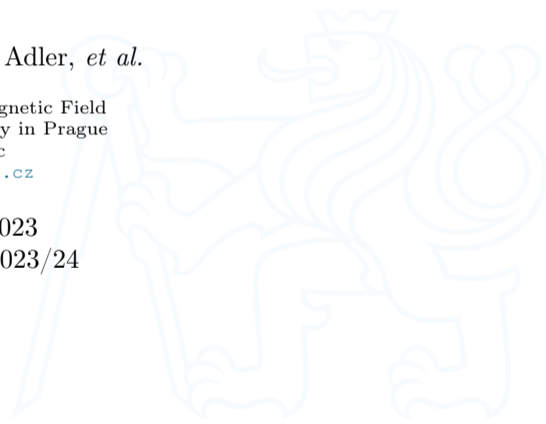
Lecture 6: Data Types: Cell, String, and Structure

B0B17MTB, BE0B17MTB – MATLAB

Miloslav Čapek, Viktor Adler, *et al.*

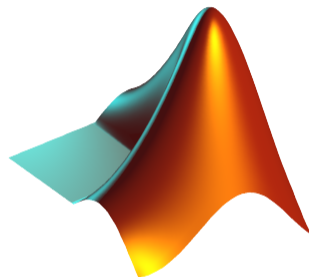
Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
matlab@fel.cvut.cz

November 6, 2023
Winter semester 2023/24





1. Cell
2. Strings
3. Structure
4. Exercises





Class cell

- ▶ Variable of class `cell` enables to store all types of variables of various dimensions (*i.e.*, for instance variable of type `cell` inside another variable of type `cell`).

- ▶ Example of a cell:

```
CL1 = {zeros(2), ones(3), rand(4), 'test', {nan(1), inf(2)}};
```

- ▶ Variable of the class `cell` can be easily allocated:

```
CL2 = cell(1, 3);
```

- ▶ Memory requirement is a trade-off for complexity of cell type.
- ▶ Typical applications of cells:
 - ▶ in `switch-case` branching for enlisting more possibilities,
 - ▶ variously long vectors of characters,
 - ▶ graphical user interface (GUI),
 - ▶ all iteration algorithms with variable size of variables,
 - ▶ packing of name-value arguments,
 - ▶ ...



Cell Indexing I.

- ▶ There are two possible ways of cell structure indexing:
 - ▶ round brackets () are used to access cells as such,
 - ▶ curly brackets { } are used to access data in individual cells.
 - ▶ Example:

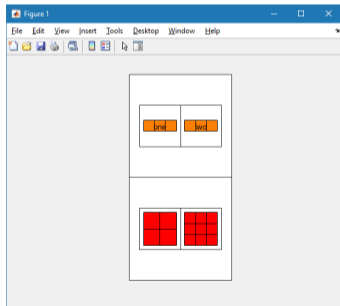
```
CL = {[1, 2; 3, 4], eye(3), 'test'};
CL(2:3)      % returns cells 2 and 3 from CL
CL{1}       % returns matrix [1 2; 3 4]
CL{1}(2, 1) % = 3

CL3 = CL(1) % CL3 is cell
M    = CL{1} % M is a matrix of numbers of type double
```



Cell Indexing II.

- ▶ Example of more complicated indexing:
- ▶ Functions to get oriented in a cell:
 - ▶ `celldisp`,
 - ▶ `cellplot`.



```
CL4 = {'one', 'two'};
CL5 = {[1, 2; 3, 4], magic(3)};
CL6 = {CL4; CL5}
CL6{2}{1}(2, 1)  % = 3
```

```
>> celldisp(CL6)
CL6{1}{1} =
one
CL6{1}{2} =
two
CL6{2}{1} =
    1    2
    3    4
CL6{2}{2} =
    8    1    6
    3    5    7
    4    9    2

>> cellplot(CL6)
```



cellfun

- ▶ Apply function to each cell in cell array
 - ▶ `A = cellfun(func, C)` returns array `A` so that `A(i) = func(C{i})`,
 - ▶ function can be defined as anonymous or a handle,
 - ▶ reduces the need for `for` loops in the code.
 - ▶ Example:

```
CL = {zeros(3), magic(4); diag(1:5), rand(6)};
nElements = cellfun(@numel, CL)
% OR:
nElements = cellfun(@(x) numel(x), CL)
nColumns = cellfun(@(x) size(x, 2), CL)
maxVal = cellfun(@(x) max(x(:)), CL)
% OR:
maxVal = cellfun(@(x) max(x, [], 'all'), CL)
```

- ▶ When returned variable cannot be concatenated into an array:

```
firstColumns = cellfun(@(x) x(:, 1), CL, 'UniformOutput', false)
```



Strings I.

- ▶ Strings in MATLAB can be represented in two forms:
 - ▶ As a vector of characters which are represented as char data type.
 - ▶ It is created using apostrophes:
- ▶ As string data type.
 - ▶ The whole sentence is string scalar.
 - ▶ It is created using double quotes:

```
st1 = 'Hello world!';
```

```
st2 = "Hello world!";
```

```
>> whos
Name      Size      Bytes  Class  Attributes
st1       1x12       24    char
st2       1x1       150   string
```

- ▶ **Distinguish between:**
 - ▶ “string” in meaning of text and
 - ▶ “string” as data type.
- ▶ Most of the functions work with both string types.
- ▶ Try to avoid diacritics (accent) in MATLAB.



Strings – Class char

- ▶ Characters are outputs of some functions (*e.g.*, `char([89, 69, 83, 33])`, `blanks(5)`).
- ▶ Each character (each element of array) requires 2 B.
- ▶ If an apostrophe is required to be part of a string, it is to be typed as two quote characters:

```
st3 = 'That''s it!'
```

- ▶ In the case of more lines of characters, it has to have same number of columns:

```
st4 = ['george'; 'pepi '];
size(st4) % [2, 6]
```

- ▶ Otherwise (usually), character arrays are stored in cell data type:

```
st5 = {'george', 'pepi', 'and all others', 'are good boys.'};
```

- ▶ Whether a given variable is of class char is tested this way:

```
ischar(st4)    % true
ischar(st5)    % false
iscellstr(st5) % true
```




Strings – Class string

- ▶ Allocation using function `strings` enables to create empty strings:

```
str1 = strings
str2 = strings(3)
str3 = strings(5, 1)
```

- ▶ Unlike `char`, `string` does not treat numbers as ASCII or Unicode.
 - ▶ When comparing `string` and `char`, type conversion is performed.

```
>> 'a' + 1
ans =
    98
```

```
>> "a" + 1
ans =
    "a1"
```

```
>> "a" < 'b'
ans =
    logical
     1
```

```
>> 'a' == 97
ans =
    logical
     1
```

```
>> "a" == 'a'
ans =
    logical
     1
```

- ▶ `strings` can be easily stored in a vector:

```
stringArray = ["a", "something", "long string"]
```

```
stringArray =
    1x3 string array
    "a"    "something"    "long string"
```

- ▶ `string` class offer many functions for a text analysis.



Strings – Type Conversion

- ▶ Quite often, it is required conversion between numbers, characters, strings and cells.
- ▶ Conversions:

```
V = [116, 101, 120, 116];
CH = 'text';
ST = "text";
CL = {'t', 'e', 'x', 't'}
```

from\to	numeric	char	string	cell
numeric	—	char(V)	string(char(V))	num2cell(char(V))
char	double(CH)	×	string(CH)	num2cell(CH)
string	double(char(ST))	char(ST)	×	num2cell(char(ST))
cell	double([CL{:}])	char(CL)'	join(string(CL), '')	×



Strings – Indexing

- ▶ Indexing in arrays of characters is the same as with a numerical arrays.
- ▶ Indexing in arrays of strings returns the whole string objects.
- ▶ Example:

```
CH1 = ['Text!!', 'string'];
CH2 = ['Hello?', 'Matlab'];
CH = [CH1; CH2];
size(CH), length(CH)
CH'
CH(1:2, 1)
CH(:)
CH(1:3:end)
```

```
ST1 = ["Text!!", "string"];
ST2 = ["Hello?", "Matlab"];
ST = [ST1; ST2]
size(ST), length(ST),
strlength(ST)
ST'
ST(1:2, 1)
ST(:)
ST(1:3:end)
```

- ▶ Indexing in strings in the same manner as in chars requires functions:

```
CH1([2:4])
CH1(8:end)
CH1(1:5)
```

=

```
extractBetween(ST1(1), 2, 4)
extractAfter(ST1(2), 1)
extractBefore(ST1(1), 6)
```



Strings – Number Conversion I. – char

- ▶ Conversion of number represented as a string (char) to number (double):

- ▶ Conversion of multiple numbers (function `str2num`):

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

```
>> str2double('-0.5453')
```

- ▶ Pay attention to possible errors:

```
>> str2num('1a')
ans =
     []
>> 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 II. – string

- ▶ Conversion of number in a string (string) to number (double):
 - ▶ Same functionality as with char:

```
>> a = "[1 2 3e-2 pi]";  
>> [num, conf] = str2num(a)  
num =  
    1.0000    2.0000    0.0300    3.1416  
conf =  
    logical  
         1  
>> str2double(a)  
ans =  
     NaN  
>> str2double("-2.35")  
ans =  
    -2.3500
```



Strings – Number Conversion III.

- ▶ Quite often is needed to convert numerical results back to a string:

```
num2str(pi);      % '3.1416'  
num2str(pi, 10); % '3.141592654'  
string(pi);      % "3.1416"
```

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

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

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

- ▶ See below...



Strings – Other Conversions

- ▶ Among others there are other functions available.

Function	Description
<code>int2str</code>	Convert integer to text. In the case the input parameter is not an integer, its value is rounded first.
<code>mat2str</code>	Converts matrix to string.
<code>hex2dec</code> , <code>dec2hex</code>	Converts hexadecimal number of type <code>char</code> to a number (and vice versa).

```
>> mat2str(magic(3))
ans =
    '[8 1 6;3 5 7;4 9 2]'
```

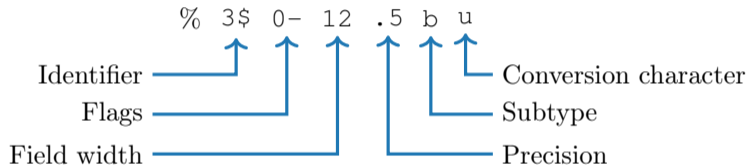
```
>> mat2str(eye(2))
ans =
    '[1 0;0 1]'
```

```
>> hex2dec('B')
ans =
    11
```



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` for black text, `fid = 2` for red text),
 - ▶ in a file (`fid` to be obtained using function `fopen`, more on later).

```
st = sprintf("The value of pi is %2.3e\n", pi);
fprintf(st)
```

```
fprintf('The value of pi is %2.3e\n', pi);
```




Strings I.

- ▶ Create following strings using `sprintf` help:

- ▶ A)

```
sprintf(..) % update the argument
```

```
ans =
```

```
'Value of pi is 3.14159, value of 5*pi is 15.70796.'
```

- ▶ B)

```
x = 50;
```

```
sprintf(..) % update the argument
```

```
ans =
```

```
'This is 50%'
```

- ▶ C)

```
tx = 'test_A';
```

```
sprintf(..) % update the argument
```

```
ans =
```

```
'This is a measurement set: test_A'
```





Strings I.

- ▶ Create following strings using `sprintf` help:

- ▶ A)

```
ans =  
    'Value of pi is 3.14159, value of 5*pi is 15.70796.'
```

- ▶ B)

```
x = 50;
```

```
ans =  
    'This is 50%'
```

- ▶ C)

```
tx = 'test_A';
```

```
ans =  
    'This is a measurement set: test_A'
```



Strings II.

- ▶ Think about differences between `disp` and `fprintf` (`sprintf`).
 - ▶ Describe the differences.
 - ▶ What function do you use in a particular situation?



Lower Case / Upper Case Characters

- ▶ Lower / upper conversion for char class:

```
st = 'RANDOMLY SIZED LETTERS';  
lower(st) % result = 'randomly sized letters'  
upper(st) % result = 'RANDOMLY SIZED LETTERS'
```

- ▶ Lower / upper conversion for string class:

```
st2 = "RANDOMLY SIZED LETTERS";  
lower(st2); st2.lower % result = "randomly sized letters"  
upper(st2); st2.upper % result = "RANDOMLY SIZED LETTERS"
```

- ▶ Support of characters from Latin 1 character set on PCs.
- ▶ Other platforms: ISO Latin-1 (ISO 8859-1).
- ▶ Supports Czech accents.



Strings – Joining

- ▶ Strings can be joined together using function `strjoin` and `join`.
 - ▶ It is applicable to variables of type `cell` and `string`.
 - ▶ Separator is optional (implicitly a space character)

```
CL = {'Once', 'upon', 'a', 'time'};
strjoin(CL) % 'Once upon a time'
strjoin(CL, '\n')
join(CL) % {'Once upon a time'}
join(CL, '\') % {'Once\upon\a\time'}
```

```
ST = ["Once", "upon", "a", "time"];
strjoin(ST) % "Once upon a time"
strjoin(ST, '\b') % backsp., "Oncupotime"
join(ST) % "Once upon a time"
join(ST, '_') % "Once_upon_a_time"
```

- ▶ Function `fullfile` connects individual inputs into a file path.
 - ▶ The separator depends on the platform (Win, Linux, Mac, ...).

```
folder1 = 'Matlab';
folder2 = 'project1';
file = 'run_process.m';
fpath = fullfile(folder1, folder2, file);
% fpath = 'Matlab\project1\run_process.m'
```



Strings – Separation I.

- ▶ 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 arbitrary.

```
this_str = 'some few little little small words';  
[token, remain] = strtok(this_str, ' ');
```



Strings – Separation II.

- ▶ Function `regexp` enables to search a string using regular expressions.
 - ▶ Syntax of the function is a bit complicated but its capabilities are vast!
 - ▶ **Example:** 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')
```

- ▶ **Example:** Search indices (positions) where words containing 'a' or 'o' begin and end.

```
[from, to] = regexp(that_str, '\w*[ao]\w*');
```

- ▶ For more details see `>> doc regexp` → Input Arguments.
- ▶ Typical tokenizer can be created in combination with above mentioned function.



Strings – Searching

- ▶ Function contains determine if pattern is in string:

```
str = ["Mary Ann Jones", "Christopher Matthew Burns", "John Paul Smith"];
contains(str, ["ann", "paul"], 'IgnoreCase', true) % logical [1 0 1]
```

- ▶ Function strfind finds a given string inside another.

- ▶ Returns indices (positions),
- ▶ searches for multiple occurrences,
- ▶ is CaSe sEnSiTiVe,
- ▶ enables to search for spaces etc.

```
str = 'This book is about history';
res = strfind(str, 'is'); % [3, 11, 21]
```

```
str2 = ["The Exakta 66 was based on the Pentacon Six but was made in West Germany.", ...
        "From January 1985 a monthly production of 200 cameras was planned."];
strfind(str2, 'was') % {[15 49]}    {[55]}
```




Strings III.

- ▶ Remove all blank spaces from the following sentence¹.

```
s = 'Do what you can, with what you have, where you are.'
```

- ▶ Try to recollect using logical indexing,
 - ▶ or use proper MATLAB function.

- ▶ Calculate how many times 'you' is used.

¹Theodore Roosevelt





Strings – Comparing I.

- ▶ Two strings can be compared using function strcmp.
 - ▶ The function is often used inside `if` or `switch` statements.
 - ▶ The result is either true or false.
 - ▶ It is possible to compare string, char and cell of strings.

```

strcmp('tel', 'A') % = 0
strcmp("tel", 'tel') % = 1
strcmp(["tel", 5], '5') % = 1
strcmp('test', {'test', 'A', '3', 6, 'test'}) % = [1, 0, 0, 0, 1]
strcmp('test', ["test", "A", "3", 6, "test"]) % = [1, 0, 0, 0, 1]
strcmp({'A', 'B'; 'C', 'D'}, ["A", "F"; "C", "C"]) % = [1, 0; 1, 0]
strcmp({'A', 'B'; 'C', 'D'}, {'A', 'F'; 'C', 'C'}) % = [1, 0; 1, 0]
  
```

`strcmp(`

A	B
C	D

`,`

A	F
C	C

`) =`

1	0
1	0



Strings – Comparing II.

- ▶ 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, n)
```

- ▶ Think about alternatives to the `strcmp` function.

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



Strings IV.

- 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 ';
strfind(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: %3.7f V\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(2, '%s a %s\n\n', B, C([1:3 5:7]));
```



Strings V.

- ▶ Write a script/function that splits following sentence into individual words using `strtok`.
 - ▶ Display number of occurrence of string `'is'`.
 - ▶ List the words individually including position of the word within the sentence (use `fprintf`).

```
sen = 'This-sentence-is-for-testing-purposes-only.';
```

```
remain = sen;  
word = 1;  
while ~isempty(remain)  
    [token, remain] = strtok(remain, '-');  
    fprintf('%2.0f. word is: %s\n', word, token);  
    word = word + 1;  
end  
fprintf('The string ''is'' is used %2.0fx.\n', length(strfind(sen, 'is')));
```



Strings VI.

- ▶ 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).

```
sen = 'This-sentence-is-for-testing-purposes-only.';
```

```
Tokens = textscan(sen, '%s', 'delimiter', '-');  
celldisp(Tokens);  
fprintf('The string 'is' is used %2.0fx.\n', length(strfind(sen, 'is')));
```



Function vs. Command Syntax

- ▶ In MATLAB exist two basic syntaxes how to call a function.

```
>> grid on % Command syntax  
>> grid('on') % Function syntax
```

```
>> disp('Hello World!') % Command syntax  
>> disp('Hello World!') % Function syntax
```

- ▶ Command syntax:

- ▶ All inputs are taken as characters.
- ▶ Outputs can't be assigned.
- ▶ Input containing spaces has to be closed in single quotation marks.

```
>> a = 1; b = 2;  
>> plus a b % = 97 + 98  
ans =  
    195  
>> p = plus a b % error  
>> p = plus(a, b);
```



Function 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 difficult to understand.
- See function help for cases where it is possible to replace `eval`.
 - **Example** storing files with serial number (`data1.mat`, `data2.mat`, ...).



String to Function, Function to String

- ▶ It is possible to construct function handle from string/character array using `str2func` function.
- ▶ The usage is in some case similar to `eval`.
 - ▶ Difference is, that `str2func` does not see variables outside the local workspace and nested functions.

```
sin = 10;
str = '@(x) sin(x)';
f1 = eval(str);
f2 = str2func(str);
```

```
f1(1)
ans =
    10
f2(1)
ans =
    0.8415
```

- ▶ Function `func2str` is used to transform function handle to character array.

```
func2str(f1)
ans =
    '@(x) sin(1)'
```



Function 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.
- ▶ Function `evalc` (“eval with capture”) serves this purpose.

```
>> CMD = evalc(['var = ' num2str(pi)])  
CMD =  
    'var = 3.1416'  
>> var  
var =  
    3.1416
```



Function `feval` – Evaluation of a Handle Function

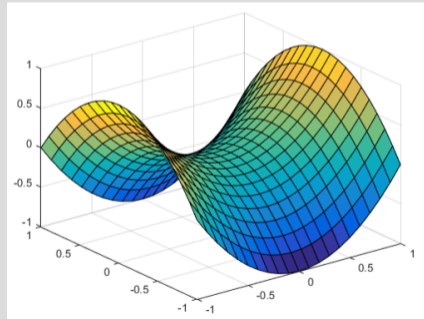
- ▶ The function is used to evaluate handle functions.
 - ▶ Simply speaking, where `eval` evaluates a string there `feval` evaluates function represented by its handle.
 - ▶ Consider this task:

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

```
hFcn = @(x,y) x.^2 - y.^2;  
X    = -1:0.1:1;  
Y    = X.';
```

```
fxY  = hFcn(X, Y);  
surf(X, Y, fxY);
```

```
fxY  = feval(hFcn, X, Y);  
surf(X, Y, fxY);
```





Function exists

- ▶ The function `exists` 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,
 - ▶ =6 p-file,
 - ▶ =2 m-file known to MATLAB (including user defined functions, if visible to MATLAB),
 - ▶ =4 mdl-file,
 - ▶ =8 class.
 - ▶ Sorted in the order of priority, returned value in bracket.

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



Reading Binary Data From a File

- ▶ Useful functions to read binary data from a file:
 - ▶ `fopen` – open the file and return the reference.
 - ▶ `fgetl` – read one line from the file, removing newline characters.
 - ▶ `fgets` – read one line from the file, keeping newline characters.
 - ▶ `feof` – test for the end of file.
 - ▶ `fclose` – close the file. **Always close the file!**

```
fid = fopen('sin.m');  
while ~feof(fid)  
    thisLine = fgetl(fid);  
    disp(thisLine);  
end  
fclose(fid);
```



Writing Data to a File

- ▶ Use `fprintf` to write a line into a file.
- ▶ It is necessary to open the file with permission for writing: `'w'`.
- ▶ Use `'\n'` to indicate new line in `fprintf` command.

```
fid = fopen('myData.txt', 'w');
D = rand(5, 3);
fprintf(fid, 'My Measured data:\n');
for iLine = 1:size(D, 1)
    fprintf(fid, '%1.4f, %1.4f, %1.4f\n', D(iLine, :));
end
fclose(fid)
```



Save Data in ASCII Format I.

- ▶ `writematrix` writes single numeric or string array variable into a file.
 - ▶ Supports `.txt`, `.dat`, `.csv`, `.xlsx`, ... formats,
 - ▶ numeric data saves in double precision,
 - ▶ multidimensional array is reshaped to 2D matrix.

```
data = reshape(1:2*3*4, 2, 3, 4);  
writematrix(data) % data.txt file created
```

```
>> type('data.txt') % show content of data.txt
```

```
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23  
2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24
```

- ▶ `readmatrix` read single array of numeric or string data.
 - ▶ Wide range of read settings (headlines, delimiters, comments, decimal separator, ...).

```
dataR = readmatrix("data.txt");  
dataR = reshape(dataR, 2, 3, 4);
```



Save Data in ASCII Format II.

- ▶ It is possible to save data in standardized ASCII format using function `save` with `'-ascii'` argument.

```
p = rand(1,5);
q = ones(3);
save('pqfile.txt','p','q','-ascii')
```

- ▶ The content of `pqfile.txt`

4.9836405e-01	9.5974396e-01	3.4038573e-01	5.8526775e-01	2.2381194e-01	
1.0000000e+00	1.0000000e+00	1.0000000e+00			variable p
1.0000000e+00	1.0000000e+00	1.0000000e+00			variable q
1.0000000e+00	1.0000000e+00	1.0000000e+00			



Variables Storing and Loading

- ▶ Existing variables in MATLAB workspace can be stored on disk.

```
>> a = 1; b = 2; c = magic(5);  
>> save % stores all variables in matlab.mat in current folder  
>> save task1 % stores all variables in task1.mat  
>> save task1 a b c % stores variables a, b and c in task1.mat
```

- ▶ The shortcut CTRL + S in Command window or Command history can be used.
- ▶ Loading variables is analogical.

```
>> load % loads matlab.mat in current folder  
>> load task1 % loads all variables from task1.mat  
>> load task1 a b c % loads variables a, b and c from task1.mat
```

- ▶ Alternatively, drag&drop the file from Current folder to Command window can be applied.



String to Function, Function to String

- ▶ Implement script/function that:
 - ▶ creates anonymous function $\mathbf{M}(x) = [\sin(x) \cos(x)]$,
 - ▶ saves this anonymous function in form of string into a text file (`*.txt`),
 - ▶ loads string from file and transforms it into anonymous function,
 - ▶ evaluates anonymous function for $x = 1$.



Indication of Running Function/Script

- ▶ How to indicate that given function/script is running?
 - ▶ Try these several possibilities ...

```
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!



Structured Variable, struct

- ▶ Data can be stored in a grouped form in structures.
- ▶ Concept is similar to OOP (without features of OOP).
- ▶ **Example:** inventory

```
stock.id = 1;  
stock.thing = "fridge";  
stock.price = 750;  
stock(2).id = 2;  
stock(2).thing = "Bowmore_12yr";  
stock(2).price = 1100;
```

- ▶ or:

```
stock = struct('id', {1, 2}, 'thing', {"fridge", "Bowmore_12yr"}, ...  
             'price', {750, 1100});
```

- ▶ Typical application: data export, complex internal variables, data in GUI, ...



Function for Works with Structures I.

▶ New field creation:

▶ Direct command.

```
stock(1).newField = 'test';
```

▶ Field name as a string.

```
setfield(stock(1), 'newField', 'test')
```

```
stock(1).('newField2') = 'test2'
```

```
stock(2).("newField3") = 'test3'
```

▶ Setting field value:

▶ Direct command.

```
stock(1).id = 3;
```

▶ Field name and value.

```
stock(1).('id') = 3;
```



Function for Works with Structures II.

- ▶ List of all fields of structure – `fieldnames`.

```
fieldnames(stock)
```

- ▶ Value of given field.

```
id2 = stock(2).id  
id2 = stock(2).('id')  
id2 = getfield(stock(2), 'id')
```

- ▶ Does given field exist?

```
isfield(stock, 'id') % = 1  
isfield(stock, 'ID') % = 0
```

- ▶ Is given variable a structure?

```
isstruct(stock) % = 1
```



Function for Works with Structures III.

- ▶ Delete field.

```
rmfield(stock, 'id')
```

- ▶ More complex indexing of structures.

- ▶ Structure may have more levels.

```
stock(1).subsection(1).order = 1  
stock(1).subsection(2).order = 2
```

- ▶ It is possible to combine cells with structures.

```
stock(1).subsection(3).check = [1; 2]  
K{1} = stock;
```

- ▶ Certain fields can be indexed using name stored as a string.

```
K{1}(1).subsection(3).('check')(2)
```



Function for Works with Structures IV.

- ▶ Getting data from fields of structure array.
 - ▶ Comma-separated list (doc [Comma-Separated Lists](#)).

```
stock.id
```

- ▶ Concatenate values to vector.

```
allIDs = [stock.id] % row vector  
allIDs = horzcat(stock.id) % row vector  
allIDs = vertcat(stock.id) % column vector
```

- ▶ Concatenate strings to cell array.

```
allThings = [stock.thing] % useless  
allThings = vertcat(stock.thing) % error  
allThings = {stock.thing} % cell array
```

- ▶ Create multiple variables.

```
allThings = {stock.thing} % cell array  
[th1, th2] = allThings{:}
```




Function for Works with Structures V.

- ▶ Set data to fields of structure array.

- ▶ `for` cycle.

```
IDs = [2, 3];  
for iStruct = 1:length(stock)  
    stock(iStruct).id = IDs(iStruct);  
end
```

- ▶ Utilizing comma-separated list.

```
IDs = {2, 3};  
[stock.id] = IDs{:};
```

- ▶ Creating multidimensional structure.

```
stock(2, 2).thing = 'multi dim.'
```

```
allThings = reshape({stock.thing}, size(stock)).'
```

Exercises



Exercise I.

- ▶ Find out how many spaces there are in the phrase “How are you?”.
 - ▶ Take a look in this lecture or MATLAB documentation and find out a suitable function.
 - ▶ Utilize logical indexing.
- ▶ Convert following string to lowercase and find number of characters.

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



Exercise II.a

- ▶ Create function that calculates volume, surface area or space diagonal of a cuboid.
 - ▶ The function accepts 4 input parameters: a, b, c and attribute, which take values 'volume', 'area' or 'diagonal'.
 - ▶ Calculate and return only an attribute required by the user.
 - ▶ Do not forget to check the input parameters.

Exercise II.b





Exercise III.a

- ▶ Create so called tokenizer (text analyzer), that
 - ▶ reads a text input `str` entered by user using function `input`,
 - ▶ reads separator `sep` (**space requires some care**),
 - ▶ split `str` into individual parts depending on `sep`,
 - ▶ store individual parts separately in a variable of type `cell`,
 - ▶ analyze how many vowels(a/e/i/y/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 III.b

► Solution using strtok.

```
str = input('Enter a text: ', 's');
sep = input('Enter separator: ', 's');
if isempty(sep) % treats all cases when sep = ' ' / ' '
    sep = ' ';
    fprintf('space ('' '') is used instead of nothing');
end

words = cell(1, 1); % allocation
nVowels = zeros(1, 1); % allocation
here = 1;
while ~isempty(str)
    [token, str] = strtok(str, sep);
    words{here} = token;
    nVowels(here) = length(regexp(token, '[aeiou]'));
    here = here + 1;
end
celldisp(words);
fprintf('%s\n', mat2str(nVowels));
```



Exercise III.c

► Solution using `strsplit`.

```
clear; clc;
str = input('Enter a text: ', 's');
sep = input('Enter separator: ', 's');
if isempty(str)
    fprintf('''string'' is empty\n');
    return;
end
if isempty(sep)
    sep = ' ';
    fprintf('space ('' ''') is used instead of nothing\n');
end

words = strsplit(str, sep);
VS = regexp(words, '[aeiou]');
nVowels = cellfun(@length, VS);

celldisp(words);
fprintf('%s\n', mat2str(nVowels));
```




Exercise IV.a

- ▶ Try to create simple unit converter, length x 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';
switch units
    case 'mm' % conversion from mm to mm (no change)
        y = x;
    case 'cm' % conversion from cm to mm
        y = 1e1*x;
    case {'in', 'inch'} % inches to mm
        y = 2.54*x;
    otherwise
        fprintf('bad units');
        return;
end
fprintf('%0.5f%s is %0.5fmm\n', x, units, y);
```





Exercise IV.b

- ▶ 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);
```

Questions?

B0B17MTB, BE0B17MTB – MATLAB
matlab@fel.cvut.cz

November 6, 2023
Winter semester 2023/24

This document has been created as a part of B(E)0B17MTB course.
Apart from educational purposes at CTU in Prague, this document may be reproduced, stored, or transmitted only with the prior permission of the authors.

Acknowledgement: Filip Kozák, Pavel Valtr, Michal Mašek, and Vít Losenický.