# Computational Methods for Electrical Engineering

## Laboratorial Worksheet 3 – Functions and strings

## 1. Objective

This laboratorial worksheet pretends to introduce the following concepts:

1. Functions.
2. String Definition and manipulation.

## 2. Functions

Besides the large number of functions that MATLAB provides in its basic configuration and in its various toolboxes, the user can create his own functions. Generally speaking, a function consists of a m-file where the function is defined according the following syntax:

```
function [var_out1, var_out2, ...] = function_name(var_in1, var_in2, ...)
% The comment lines between the reserved word 'function' and the code of the function
% are used as the help text whenever the command 'help' is used in the command window
...
%% code %%
...
end
```

There are some of the rules that should be considered when creating a function:

1. One file per function created.
2. It is strongly recommended that the file name and the function name are the same. If this is not verified, MATLAB will consider the file name instead of the function name when invoking the function, a situation that can lead to some confusion.
3. The first instruction of the function must be the reserved word **`function`**. You can, however, write comment lines before the reserved word function. In this case, this is the text that is used when calling the function's help.

Consider the following example of the **`media.m`** function, which returns the average of the values contained in a vector:

```
function y = media(x)
if ~isvector(x)                                % it checks if x is a vector
    error('Input value must be a vector') % it returns an error message
end
y = sum(x)/length(x);
end
```

These are some examples of usage of this function in the command window:

```
>> media(10)
ans =
    10
>> media(1:10)
ans =
    5.5000
>> media([1 2; 3 4])
Error using media
Input value must be a vector
```

As you can see, the function uses the **`isvector()`**[1] function to verify if the input variable is a vector. If this is not the case, it returns an error message via the **`error()`** function, which forces the end of the function.

## 3. Strings

### 3.1 Creating a String

A char vector, usually referred as string in the literature, consists of characters delimited by single quotation marks. The following instructions present a few examples for creating some strings:

```
>> str_empty = '';          % it creates an empty string (only single quotation marks!!)
>> str1 = 'Hello world';    % well, this example had to be here, right?
>> str2 = 'MCEE 2022/2023'; % another classic, of course
>> str3 = '0303456';        % string containing only numeric chars
```

Another way to assign a string to a variable can be achieved using the **`input()`** function, as shown in the previous worksheet:

```
>> nome = input('Your name, please: ', 's'); % don’t Forget the second argument, 's'
```

These variables can be operated as any other variable in MATLAB (as far as the operations make sense, of course):

```
>> length(str_empty)
ans =
     0

>> length(str1)
ans =
    11

>> sub_str2 = str2(1:4)
sub_str2 =
    'MCEE'
```

It is also possible to define a matrix of strings. However, you should not forget that a string, is a char vector, so every row of the matrix must contain the same number of columns, i.e., each line must have the same number of chars:

```
>> M = ['MCEE'; '2022'; '2023']
M =
  3×4 char array
```

[1] **`isvector`** belongs to a set of functions referred as *detect state functions*. In the command window, execute **`doc is*`** for obtaining further information. Among these function, you can find **`ischar`**, **`isempty`**, **`isscalar`** , **`ismatrix`** , and **`isnumeric`**, just to list a few more commonly used.

```
        'MCEE'
        '2022'
        '2023'
    >> size (M)
    ans =
         3     4

    >> N = ['short'; 'Supercalifragilisticexpialidocious']
    Error using vertcat
    Dimensions of arrays being concatenated are not consistent.
```

## 3.2 Operations with strings

String **concatenation** consists of joining two or more strings into a single string. This can be achieved by defining a new vector that includes the desired strings, or using the **strcat()** function:

```
    >> str1 = 'Beginning  '; % two added spaces at the end
    >> str2 = '   End';      % three added spaces at the beginning
    >> v1 = [str1, str2]
    v1 =
        'Beginning     End'
    >> length(v1)
    ans =
        17

    >> v2 = strcat(str1, str2)
    v2 =
        'Beginning   End'

    >> length(v2)
    ans =
        15
```

There is a difference between the way the strings were concatenated in the previous example. In the first one, all spaces either at the beginning or at the end of each string are included in the concatenated string. In the second one, all spaces at the end of each string are removed. This can be clearly observed in the next example:

```
    >> s1 = 'MCEE    ';   % note the additional spaces at the end...
    >> s2 = ' 2022    ';   % ...of every string
    >> s3 = '/2023    ';
    >> strcat(s1, s2, s3)
    ans =
        'MCEE 2022/2023'
```

The **sprintf** function converts formatted data into an array of chars. In other words, it operates the same way as **fprintf** function, introduced in the previous worksheet, but instead of displaying a message on the screen, it generates a string. All formatting options considered for **fprintf** function are valid for **sprintf**:

```
    >> t = 0.1;
    >> q = 15.32;
    >> msg1 = sprintf(' Capacitor charge, q(t = %.1f) = %.2f', t, q)
    msg1 =
        'Capacitor charge, q(t = 0.1) = 15.32'
```

There are also functions that convert the characters of a string into uppercase, **upper()**, and lowercase, **lower()**:

```
>> str = ' LoOk A sTRinG!';
>> lower(str)
ans =
    ' look a string!'

>> upper(str)
ans =
    ' LOOK A STRING!'
```

## 3.3 Strings comparison

When comparing strings, a very common pitfall is using the relational operator "**==**". It is important to note that, as explained in the previous worksheet, this is an element-wise operation, i.e., the output will be a vector of logical values:

```
>> str1 = 'gato';
>> str2 = 'rato';
>> str3 = 'GATO';
>> str1 == str2
ans =
  1×4 logical array
   0   1   1   1

>> str1 == str3
ans =
  1×4 logical array
   0   0   0   0
```

If the goal is to find out if both strings are identical (*true*) or different (*false*), functions **strcmp()** or **strcmpi()** should be considered. **strcmp()** compares two strings and they are considered identical if the size and content of each are the same, while **strcmpi()** performs the same way ignoring case:

```
>> strcmp(str1, str2)
ans =
  logical
   0

>> strcmp(str1, str3)
ans =
  logical
   0

>> strcmpi(str1, str3) % ignores case
ans =
  logical
   1
```

To split a string into sub-strings, we can use the **strtok** function. This function looks for the first delimiter character of the string, i.e., the character that defines the point where the string will be split, returning two string: the first string is obtained from the original string up to the delimiter character, not included; the second string consists of the remaining of the original string, including the delimiter character. By default, the delimiter character is the space character. However, the user can define a different character. If the delimiter character is at the beginning of the string, it is ignored:

```
>> str = '  Universidade de Coimbra';
>> [str1, resto] = strtok(str)
str1 =
    'Universidade'
resto =
    ' de Coimbra'

>> [str2, resto] = strtok(resto)
str2 =
    'de'
resto =
    ' Coimbra'

>> [str3, resto] = strtok(resto)
str3 =
    'Coimbra'
resto =
  0×0 empty char array
```

Finally, it is presented a short table which includes a brief description of the most commonly functions used in string manipulation, besides those explained in this section.

Also, a final comment to fact that, since version R2016b, there is also available a **string data type**. The concept of strings discussed in this section are actually arrays of type **char**, i.e., each array element is just one character. In the string type, each array element is, in fact, a string. As the **string data type** will not be studied in depth in this course, this matter will not be explored in this worksheet.

| Function | Description |
|---|---|
| **`blanks(`***n***`)`** | Create character array of *n* blanks. |
| **`count(`***str, sub_str***`)`** | Count occurrences of pattern *sub_str* in string *str*. |
| **`deblank(`***str***`)`** | Remove trailing whitespace from end of string *str*. |
| **`erase(`***str, sub_str***`)`** | Delete substring *sub_str* within string *str*. |
| **`int2str(`***n***`)`** | Convert the integer *n* to a character array that represents the integer. If *n* contains floating-point values, this value is rounded before conversion. |
| **`isletter(`***str***`)`** | Determine which characters are letters in *str*. |
| **`ispace(`***str***`)`** | Determine which characters are space characters in *str*. |
| **`num2str(`***n***`)`** | Convert a number *n* to a char vector. |
| **`str2num(`***str***`)`** | Convert a character array *str* to numeric array. If *str* contains non-numeric chars, it returns an empty array. |
| **`strncmp(`***s1, s2, n***`)`** **`strncmpi(`***s1, s2, n***`)`** | Compare first *n* characters of *s1* and *s2*. **`strncmpi()`** is non-case sensitive. |
| **`strfind(`***str, sub_str***`)`** | Return the starting index of substring *sub_str* included in *str*. |
| **`strrep(`***str, old, new***`)`** | Replace all occurrences of *old* in *str* with *new.* |
| **`strtrim(`***str***`)`** | Remove leading and trailing whitespace from *str*. |

## 4. Activities

### 4.1 Activity 1 – Introduction

In this first activity, it is intended to implement the function below presented, and verify its operation through the command window.

a) Create a function containing the following instructions (don't forget that **the file must have the same name as the function** and ignore the line numbering):

```
function [V, P] = voltpower(R, I)
% The voltpower function returns the voltage across a resistor R
% and the dissipated power considering an electrical current I
%
% Use:
%     [V, P] = voltpower(R, I)
%      V - voltage across R (V)
%      P - dissipated power in R (W)
%      R - electrical resistance, R >= 0 Ohms (scalar)
%      I - electric current (A) (vector)
V = R * I;
P = R * I.^2;
```

b) Invoke the function in the command window considering the following instructions, and check if it is working as expected:

```
1. >> voltpower(5, 2)

2. >> V = voltpower(5, 2)

3. >> [V, P] = voltpower(5, 2)

4. >> [V1, P1] = voltpower(5, 1:5)

5. >> help voltpower
```

c) As you can see in the comment lines, the first argument (**R**) of the function must be a scalar equal to or greater than zero. Edit the voltpower function so that an error is displayed case **R** that is not a scalar, i.e., R cannot have more than one row and one column (*hint: don't forget the detect state functions available in MATLAB, referred to on page 2*). Call the **`error()`** function to return an error message and end the function **`voltpower`**. The function should now behave similar to the following example (note that the error line may vary depending on the code you write):

```
>> [V, P] = voltpower([1 2 3],2)
Error using voltpower
R must be a scalar
```

d) Edit again the voltpower function, this time to return an error case **R** is a negative value:

```
>> [V, P] = voltpower(-5,2)
Error using voltpower
R must be equal to or greater than zero
```

e) Finally, edit the function in order to verify that the second argument is a vector, i.e., it does not have more than one row or one column. For this, consider using one of the detect state functions to test whether or not **I** is a vector:

```
>> [V, P] = voltpower(5,[1 2; 3 4])
Error using voltpower
I must be a scalar or a vector
```

## 4.2 Activity 2

Write the following functions:

a) **`x = inch2cm(num)`**: convert a value in inches (*num*) to centimetres (1 inch = 2.54 cm).

```
>> inch2cm(3)
ans =
    7.6200
```

b) **`[v1, v2] = posneg(v)`**: takes a vector of values, **v** (row or column) and returns two vectors: one containing all elements of **v** greater than or equal to zero; and another containing all the negative elements of **v**. The function must also return an error message if v is not a vector (but it can be a scalar).

```
>> [x, y] = posneg([1 -2 3 -4 5 -6 7])
x =
     1     3     5     7
y =
    -2    -4    -6
```

c) **`v = preenchedir(str, n)`**: this function fills the right end of a string *str* with black chars until its length is equal to *n*. If *str* is longer than *n*, the function returns only the first *n* chars of *str* (*str is* truncated).

```
>> preenchedir('Wasserflasche', 20)
ans =
    'Wasserflasche       '

>> preenchedir('Wasserflasche', 6)
ans =
    'Wasser'

>> preenchedir('Wasserflasche', 0)
Error using preenchedir
n must be equal to or greater than 1

>> preenchedir(['MCEE'; '2023'], 3)
Error using preenchedir
str must be a char row vector
```

## 4.3 Activity 3

A simple way to encrypt messages is to use the first letter of each word in a sentence to hide the message to be encoded. Write a script named **`activity3.m`** that asks the user for the sentence to be decoded, and that returns the decoded message.

To decode the message, write the function **`decode(str)`**, which receives the sentence containing the message to be decoded and which returns the decoded message. Assume that each word is separated by only one space character.

Consider the following example of presentation for the script to develop:

```
Activity 3
Message decoder

Sentence to decode: Few risks are greater in life, enjoy!
Decoded message: Fragile
```

*Hint 1: Consider that the number of words in the message is equal to the number of spaces plus one. Note that the* **count** *function allows you to count the number of times a given pattern is presented in a string..*

*Hint 2: Use the function* **strtok** *to move to the next word.*

## 5. Out of the laboratory

### 5.1 Exercise 1

Write the function **[r, t, f] = CoordEsf(x, y, z)** which received the cartesian coordinates **x**, **y**, and **z**, and returns the corresponding spherical coordinates. **x**, **y**, and **z** can be vectors, which means that they all must have the same number of elements. The function should return an error message if **x**, **y**, and **z** are not vectors. Also, it should return an error message if the vectors do not have the same number of elements. $\theta$ and $\phi$ values are returned in degrees.

$$r = \sqrt{x^2 + y^2 + z^2}, \quad \theta = \arccos\left(\frac{z}{r}\right), \quad \phi = \arctan\left(\frac{y}{x}\right)$$

```
>> help CoordEsf
   CoordEsf transforms cartesian coordinates into
   spherical coordinates.
     [r, theta, phi] = CoordEsf(x, y, z)
     x, y, e z must be vectors with the same length
     theta e phi are obtained in degrees.

>> [a, b, c] = CoordEsf(0, 2, 3)
a =
    3.6056
b =
   33.6901
c =
    90

>> [a, b, c] = CoordEsf([10 20], [5 0], [-4 2])
a =
   11.8743   20.0998
b =
  109.6857   84.2894
c =
   26.5651         0
```

### 5.2 Exercise 2

Write the function **B = swapcol(A, i, j)** which returns the matrix **A** with columns **i** and **j** swapped. The matrix **A** must have at least two columns and the arguments **i** and **j** should not have values greater than the number of columns in **A**. Failure to comply with any of these specifications returns an error message.

```
>> help swapcol
```

```
            swapcol returns A with columns i and j swapped
              M = swapcol(A, i, j)
              A is a matrix with at least two columns
              i e j must be equal to or less than the number os columns of A

          >> M = [10  40  70  100;  20  50  80  110; 30  60  90  120]
          M =
              10    40    70   100
              20    50    80   110
              30    60    90   120

          >> N = swapcol (M, 2, 4)
          N =
              10   100    70    40
              20   110    80    50
              30   120    90    60

          >> swapcol(M, 2, 6)
          Error using swapcol
          i e j devem ser iguais ou inferiores ao número de colunas de A
```

## 5.3 Exercise 3

Write a function **`CheckerBoard`** which receives a integer value *n*, and returns a matrix $M_{nxn}$ with the following pattern (e.g., *n* = 5):

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Note that the element M(1, 1) is always the value 1, for any value of *n*, considering that *n* must be a positive integer.

*Hint: This exercise can be solved in many different ways. One way could be to create a vector for the odd lines and another for the even lines, which can later be replicated with the* **`repeat`** *function. Another alternative could be taking chance of the fact that the several values of '1' (or the '0') are arranged along diagonals. You can provide some sort of algorithm that allows you to generate the necessary diagonals, using the* **`diag`** *function.*

```
     >> M = CheckerBoard(1)
     M =
          1

     >> M = CheckerBoard(2)
     M =
          1     0
          0     1
     >> M = CheckerBoard(3)
     M =
          1     0     1
          0     1     0
          1     0     1
```

### 5.4 Exercise 4

From the function written in activity 3, write a new function, **decode2**, which receives a string that may contain more than one space between words. This function must, before decoding the message contained in the string, delete the additional spaces and then decode the message.

```
>> [msg, clean] = decode2('Few  risks    are   greater     in  life,     enjoy!')
msg =
    'Fragile'
clean =
    'Few risks are greater in life, enjoy!'
```

### 5.5 Exercise 5

The board game "O Homem que Mordeu o Cão[2]" allows the creation of some disconcerting phrases obtained by the random selection of the components that constitute their structure. It is intended to write a script that randomly composes a sentence with the following structure:

"[*Where*], [*Who*] [*What*] [*To Whom*] because of [*Cause*]"

For this script, the char matrices **Where**, **Who**, **What**, **ToWhom** e **Cause** must be created, and they must have the following lines of text (you can add your own ideas, of course!):

*Where = {'At FCTUC', 'In Coimbra', 'At Rua Larga', 'At Largo da Portagem'}*

*Who = {'a man', 'a woman', 'a dog', 'a thief'}*

*What = {'spoke to', 'yelled at', 'surprised', 'bit'}*

*ToWhom = {'a police officer', 'a baker', 'a lettuce', 'a polar bear'}*

*Cause = {'a sleepless night', 'a hard exam', 'the gas price', 'Benfica's performance'}*

It is important to note that the proposed lines of text have different lengths, so it is recommended to use the function **char** to create each of the char matrices.

The script must select randomly a row of each matrix, and concatenate these selected strings accordingly to the pattern proposed. After displaying the sentence, the script should ask if the user pretends to repeat. The user can enter an 'y' or just press <enter> to repeat. Any other answer will terminate the script.

```
O Homem que mordeu o cão
Press a key to continue...

At Largo da Portagem, a man yelled at a baker because of Benfica's performance.
Again? [Y/n]: y
At FCTUC, a thief surprised a lettuce because of a sleepless night.
Again? [S/n]:
In Coimbra, a man bit a polar bear because of a hard exam.
Again? [S/n]: n
```

*Hint 1: You can use the function* **pause** *to freeze the script until the user hit a key.*

*Hint 2: Consider using the function* **isempty** *for detecting whenever the user just hits the <enter> key.*

*Hint 3: In case you use function* ***strcat*** *to concatenate the sentence, you may consider using ASCII value 32 for introducing a space char (****strcat(****str1,* ***32****, str2****)****).*

[2] "O Homem que Mordeu o Cão" (The man who bit the dog) is a Portuguese humorous radio segment about unusual histories, that later originated a game board.

## 5.6 Exercise 6

Write a script **`rps.m`** that reproduces the popular game "Rock Paper Scissors". The user selects an option using the keys 'r' (rock), 'p' (paper) and 's' (scissors). The game runs continuously until the user hits <enter> without any option. For each match, the script should present the player selection, the computer selection, the result of the match, and the global score of the game.

```
[R]ock, [P]aper, [S]cissors? (<enter> to end): r <enter>
Player: Rock - Computer: Rock
Tie!

Score:
   Player: 0
   Computer: 0

[R]ock, [P]aper, [S]cissors? (<enter> to end): p <enter>
Player: Paper - Computer: Rock
You won!

Score:
   Player: 1
   Computer: 0

[R]ock, [P]aper, [S]cissors? (<enter> to end): <enter>

>>
```