Functions in MATLAB

Lecture 3

Dr. Görkem Saygılı

Department of Biomedical Engineering
Ankara University

Introduction to MATLAB, 2017-2018 Spring

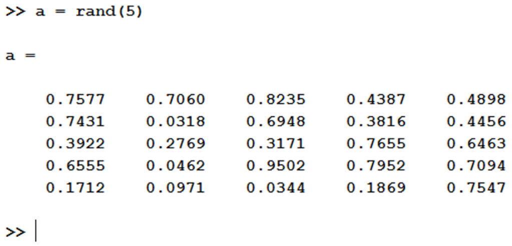
Writing Functions in MATLAB:

We already see that we can use Command Window to enter commands in MATLAB. The problem with using Command Window is that it is not possible to load a sequence of previous commands at once easily. We need functions to easily re-run our previous code.In MATLAB, we can also write functions so that we can use them in other scripts.

We have already seen some built-in functions in MATLAB. In this lecture, we will see how to create our own functions.

rand() Function:

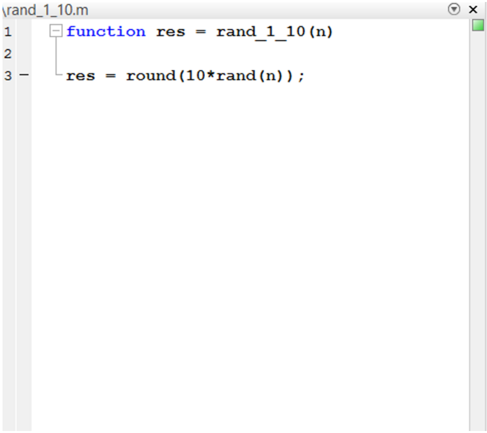
rand() is function in MATLAB to generate random variables between 0 to 1. rand(n) generates an $n \times n$ matrix consisting the generated random variables:



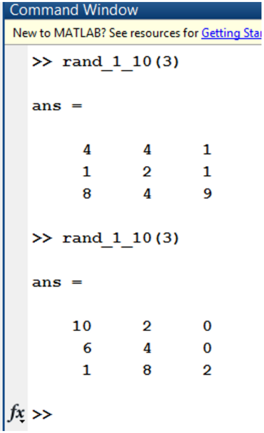
```
>> a = rand(5)

a =

    0.7577    0.7060    0.8235    0.4387    0.4898
    0.7431    0.0318    0.6948    0.3816    0.4456
    0.3922    0.2769    0.3171    0.7655    0.6463
    0.6555    0.0462    0.9502    0.7952    0.7094
    0.1712    0.0971    0.0344    0.1869    0.7547

>> |
```

Generating Numbers Between 0 and 10:

Let's write a function so that it generates integer numbers between 0 to 10:



```
function res = rand_1_10(n)

res = round(10*rand(n));
```



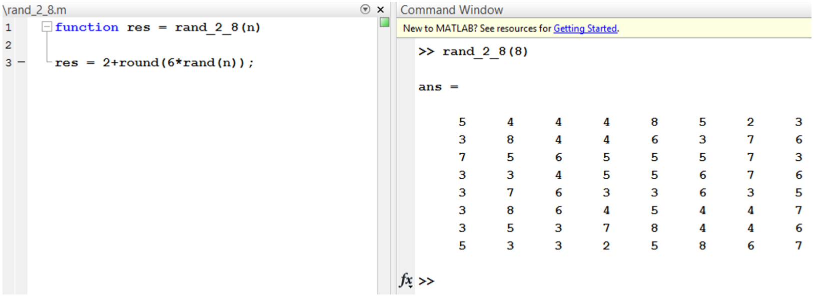
```
>> rand_1_10(3)

ans =

     4     4     1
     1     2     1
     8     4     9

>> rand_1_10(3)

ans =

    10     2     0
     6     4     0
     1     8     2

fx >>
```

Exercise:

Similarly, can you code a function that can generate $n \times n$ random integers between 2 and 8?

Solution:

We already know how to generate $n \times n$ random integers from 0 to 6. If we further increment the result by 2, we would end up with random integers from 2 to 8:



\rand_2_8.m

```
function res = rand_2_8(n)

res = 2+round(6*rand(n));
```

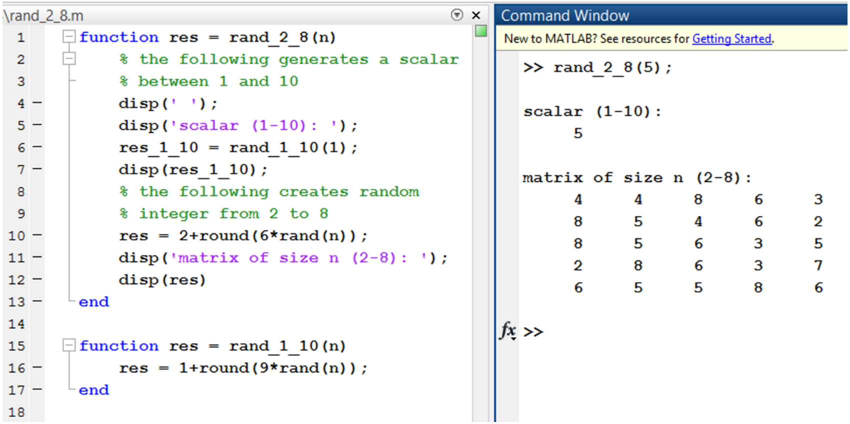
Command Window

New to MATLAB? See resources for Getting Started.

```
>> rand_2_8(8)

ans =

     5     4     4     4     8     5     2     3
     3     8     4     4     6     3     7     6
     7     5     6     5     5     5     7     3
     3     3     4     5     5     6     7     6
     3     7     6     3     3     6     3     5
     3     8     6     4     5     4     4     7
     3     5     3     7     8     4     4     6
     5     3     3     2     5     8     6     7

>>
```

End Keyword for Functions:

You can use end keyword to indicate the end of a function. But it is not very common except you use multiple functions:



\rand_2_8.m

```matlab
function res = rand_2_8(n)
    % the following generates a scalar
    % between 1 and 10
    disp(' ');
    disp('scalar (1-10): ');
    res_1_10 = rand_1_10(1);
    disp(res_1_10);
    % the following creates random
    % integer from 2 to 8
    res = 2+round(6*rand(n));
    disp('matrix of size n (2-8): ');
    disp(res)
end

function res = rand_1_10(n)
    res = 1+round(9*rand(n));
end
```

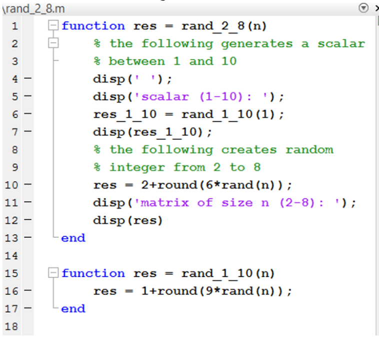
Command Window

New to MATLAB? See resources for Getting Started.

```
>> rand_2_8(5);

scalar (1-10):
     5

matrix of size n (2-8):
     4     4     8     6     3
     8     5     4     6     2
     8     5     6     3     5
     2     8     6     3     7
     6     5     5     8     6

fx >>
```

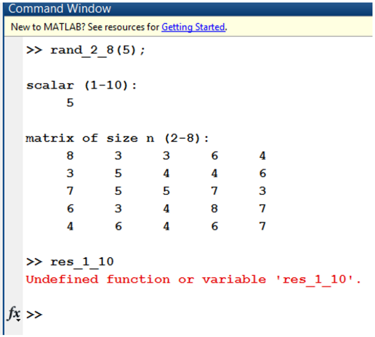
Variable Declarations in a Function (Local Variables):

In MATLAB, the variables declared in a function can not be accessed from Command Window afterwards. They are local. These variables are just for the use inside a function:



\rand_2_8.m

```
function res = rand_2_8(n)
    % the following generates a scalar
    % between 1 and 10
    disp(' ');
    disp('scalar (1-10): ');
    res_1_10 = rand_1_10(1);
    disp(res_1_10);
    % the following creates random
    % integer from 2 to 8
    res = 2+round(6*rand(n));
    disp('matrix of size n (2-8): ');
    disp(res)
end

function res = rand_1_10(n)
    res = 1+round(9*rand(n));
end
```



Command Window

New to MATLAB? See resources for Getting Started.

```
>> rand_2_8(5);

scalar (1-10):
     5

matrix of size n (2-8):
     8     3     3     6     4
     3     5     4     4     6
     7     5     5     7     3
     6     3     4     8     7
     4     6     4     6     7

>> res_1_10
Undefined function or variable 'res_1_10'.

>>
```

The Reason of Function-Scoped Variables:

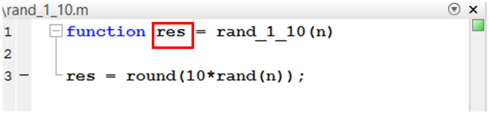
If the variables inside a function changes the values of variables in the workspace, then the workspace becomes messy with so many variables from different functions.

If function variables were the same as workspace variables, the user would have to keep track of the changes for each variable previously defined in the workspace and possibly could had been modified by the function.

The situation would become even more complex, when you are using multiple functions together (nested functions).

How to Return the Resulting Variable from a Function:

We do our calculations using functions and at the end, the variables declared inside the function is not accessible from outside of the function. The way to access these variables after the execution of the function is to indicate them as output variables:

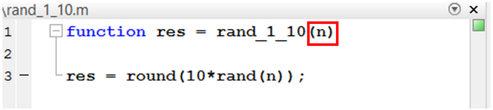

\rand_1_10.m

```
function res = rand_1_10(n)

res = round(10*rand(n));
```

In red, the output variable of the function is indicated.

Function Input:

Function name following by parenthesis, we can define our inputs to the function:



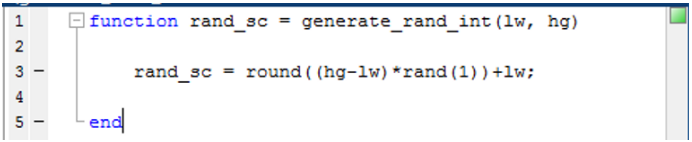
```
function res = rand_1_10(n)

res = round(10*rand(n));
```

The inputs are separated using commas in between.

Exercise:

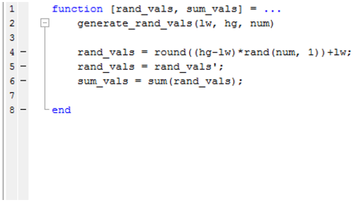
Let's write a MATLAB function that accepts a minimum and maximum threshold and returns an integer scalar between these thresholds as output.

Solution:

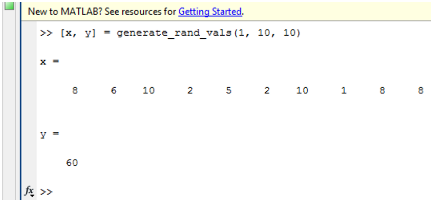

```
function rand_sc = generate_rand_int(lw, hg)

    rand_sc = round((hg-lw)*rand(1))+lw;

end
```

Multiple Outputs:

Let's say we want to generate 10 integers between 0 to 10 and we want to also calculate the sum of the generated sequence of numbers. We do not need to write two separate functions. We can instead return both results using a single function with multiple outputs:



```
function [rand_vals, sum_vals] = ...
    generate_rand_vals(lw, hg, num)

    rand_vals = round((hg-lw)*rand(num, 1))+lw;
    rand_vals = rand_vals';
    sum_vals = sum(rand_vals);

end
```

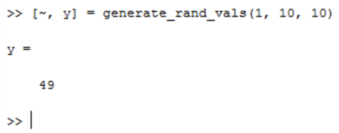


New to MATLAB? See resources for Getting Started.

```
>> [x, y] = generate_rand_vals(1, 10, 10)

x =

     8     6    10     2     5     2    10     1     8     8


y =

    60

>>
```

Using The Output of Interest:

Functions may produce multiple outputs, yet we might not need to use all of them at each time. Consider our previous example and imagine we only need the sum of the generated variables:



```
>> [~, y] = generate_rand_vals(1, 10, 10)

y =

    49

>> |
```

Formal Definition of Function in MATLAB:

As common for all programming languages, we need to follow some rules while creating functions in MATLAB:

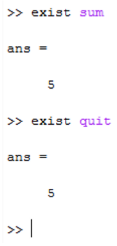
utput2, ... = functionName(input1, input2, ...).

- We had some syntactic rules while defining variable names, same applies to function names.
- Although not a must, the function name should be the same as the file name of the function (functionName.m). If they are different, MATLAB just considers the name of the .m file.
- The function declaration must be at the first line. Only exceptions are the comments.
- Avoid using the same names as MATLAB built-in functions for your functions. If you do, MATLAB will execute your function (if exists in its path) than the built-in function.

Exist Built-in Function:

How can we be sure if there is a variable or built-in function that has the same name as the name we want to use?

Luckily we have a built-in function to check if the name is already in use: exist.



```
>> exist sum

ans =

     5

>> exist quit

ans =

     5

>> 
```

Subfunctions:

We can write multiple functions in a M-file as we already saw. The first function is called the main function and the other functions are subfunctions.

We can write as many subfunctions as we want.

The use of keyword end is optional. Yet, if you use it once, you need to use it for all functions inside the .m file. Otherwise, you can omit end keyword for all functions inside the file.

The Scope of Variables:

The scope of a variable defines the region in a code where the variable is accessible. For example, a declared variable inside a function is valid for the statements of that function after the declaration.

Whenever execution of a function is finished, all the variables initialized in that function are deleted and no longer exists in the workspace after MATLAB returns back to the Command Window.

Such variables are called local variables.

Global Scope Variables:

If you want a variable to be accessible from multiple functions and from Command Window, you can define it as a global variable.

Any changes to a global variable effects all other functions using that variable afterwards. Hence, you need to be careful while using global variables.

Global variables are especially common in MATLAB GUI programs.