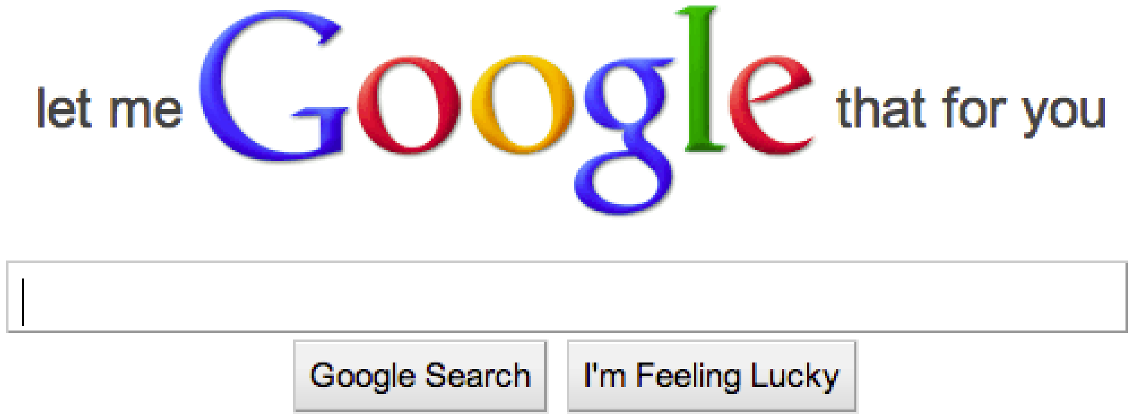
Matlab Basics

Yaara Erez
yaara.erez@mrc-cbu.cam.ac.uk

MRC Cognition and Brain Sciences Unit
November 2017

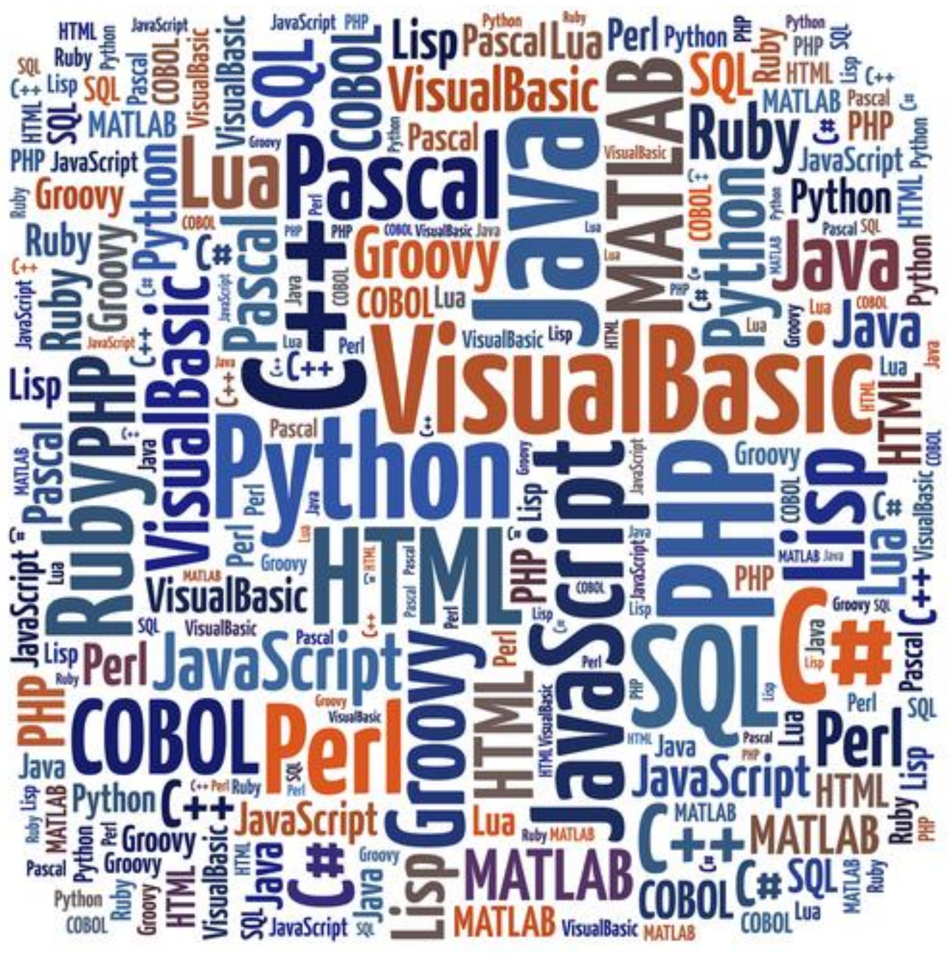
Some good news



Top Tip

Focus on the concepts, not the details… and google everything else

Some more good news



Talk outline

- Code files (scripts, functions)
- Data files
- Flow control
  - Conditioning – if, switch
  - Repetitions: for loop, while loop

These will be demonstrated in the extra scripts only

- More examples are in the scripts on the wiki.

Code files

- Instead of writing commands in the prompt, we can write them in a code file and then execute (run) them as many times as we want.
- Code files are files with **extension ‘.m’**.
- Code files can be either (batch) **scripts** or **functions**.
- Can be opened and edited in the Matlab editor (or other editors).

Script basics

A script is a list of commands that are executed almost as if you were typing them into the command window, line by line

Action:

- Open a new script
- Create a variable, x, which is a list of 5 numbers
- Save it as matlab_basics
- Run script ▶

Script basics

- ‘%’ for bits you don’t want to be run (titles, notes etc.)

Add me

Top Tip

Use these liberally!

- ‘;’ To stop line printing (echo) in command window

Add me

- save(‘filename’,’variables’)

Add me

```
save(‘test.mat’, ‘x’)
```

- F5-run script, F9-run highlighted bit

Run save

Data files

- Any Matlab variable(s) can be saved in a data file.
- Matlab data files have ‘.mat’ extension.
- ‘save’: save variables into a mat-file.
  - `save(‘file_name’)` → save all variables to file_name
  - `save(‘file_name’, ‘var1_name’, ‘var2_name’)` → save only some of the variables into file_name.
    - Note: var1_name, var2_name, etc. should be strings.

- ‘load’: load variables from a mat-file into the workspace.
  - `load(‘file_name’)` → load all variables in file_name
  - Can also specify the names of the variables that needs to be loaded.

Flow control

- Generally, in a script/function, commands are executed line by line, from start to end.
- But there are several special commands that change that order.
  - Conditioning: only execute something under certain conditions (if, switch)
  - Repetition: repeat a command or a series of commands (for, while loops).

If

```
if this is true
     %Do whatever is in the middle
elseif this is true
     %Do whatever is in the middle
else
     %Do whatever is in the middle if
     neither above are true
end
```

If

```
a = 33;

if a < 30
    disp('small')
elseif a < 80
    disp('medium')
else
    disp('large')
end
```

Comparison operators

- Operators that tell us how two variables relate

Type 2>3 and run

- 1= true, 0 = false

- Can run on lists, 2D data and... any dimension of data

Type and run:

```
a = randi(100, 10)
a>= 50
```

Operator	Meaning
==	equal to
~=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Combining Operators

Operator	Meaning
~	NOT/OPPOSITE
&	AND (need true AND true)
|	OR (need true OR true)
==	equal to
~=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

y = 5

y > 3| y ~= 5

True or False

= 1

Operator	Meaning
& (or && for logical scalars)	AND
| (or || for logical scalars)	OR

What would be the answer to:

x = 8

y = 9

~(~(x < 3))&~(y >14 | y>10)

Create an If statement

- x = 10, minVal = 2, maxVal = 6
- Write a script to print out (using ‘disp’):

a) ‘Value within range’ if x is within or equal to the range parameters
b) ‘Value exceeds maximum value’ if it’s larger than maxVal
c) ‘Value is below minimum value’ if it’s smaller than minVal
d) Test different x to check it’s working

Answer

```
x = 10;
minVal = 2;
maxVal = 6;

if (x >= minVal) && (x <= maxVal)
    disp('Value within specified range.')
elseif (x > maxVal)
    disp('Value exceeds maximum value.')
else
    disp('Value is below minimum value.')
end
```

Repetitions: For loops

```
%General structure:
for index = values
     %Do whatever is in the middle
end
```

```
%Example:
data = [1 : 100];
n = length(data);
result = 0;
for k = 1 : n
     result = result + data(k);
end
result_2 = result/n
```

Top Tip

Use variable names that describe what it is

Create a for loop

- Define an array with 5 numbers between 0 to 10 as you like. Each number represents the score of a subject in a test.
- For each subject, apply a correcting factor on the grades. Create a new variable which will contain the revised grades. The factor should be:

  $x = x*1.2$

- If the revised grade is larger than 10, set it to 10.
- In the workspace, make sure you can see the two variables and that their values make sense.

Answer

```
score = [1, 5, 7, 9, 8];
n = length(score);
for ind = 1:n
    revised_score(ind) = score(ind)*1.2;
        if revised_score(ind) > 10
            revised_score(ind) = 10;
        end
end
```

Top Tip

Initialize arrays rather than growing with each loop
E.g. use revised_scores= zeros(size(score))

Top Tip

Use 'size' function instead of 'length' so you can make sure your loop runs on the correct dimension

Another answer

Use arithmetic operations instead of loops wherever possible – it's faster!

```
score = [1, 5, 7, 9, 8];
revised_score = score *1.2;
revised_score (revised_score > 10) = 10;
```

Other more efficient solutions instead of loops:

- 'find' + length' or 'sum
- 'isequal', 'isempty', 'all', 'any'

Functions

- You can run a script from the command line or from another script

- Put your for loop in a new script and save as my_for_loop
- Run your script by typing my_for_loop into the command window

- Want more flexibility and more encapsulation? Functions…
- Similar to a script but you pass input values and return output values

Functions

```
function [outputs] = function_name(inputs)

%Put your script in here

end
```

Save the script as ‘function_name’

Scripts vs. functions

Script

- Exactly the same as running commands in the prompt
- Variables are recognized in the ***global*** workspace
- No input/output arguments
- Execute: F5 for all, or highlight and F9

Function

- An encapsulated piece of code with a ***local*** workspace (scope)
- Variables are not recognized in the global workspace
- Input/output arguments
- Can be general (applies on any data, project)
- First line of code MUST BE:

```
Function [<out_arg>] = <function_name>(<in_arg>)
```

Better to use functions whenever you can

(in my opinion)

To avoid any confusion of variable names and content

- You may start writing a batch script, then later convert section of it into functions

Create a function

- Want to revise score with any given factor (variable called ‘correct_factor’), not just *1.2
- Turn your for loop script into a function that takes inputs: ‘scores’ and ‘correct_factor’ and returns the revised scores as an output
- Run from the command line with a few different inputs to test