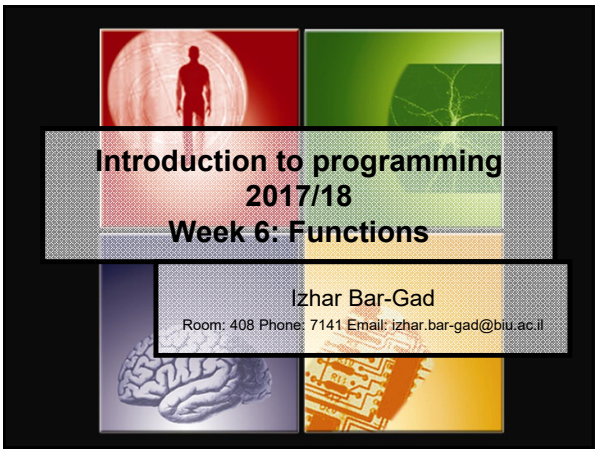
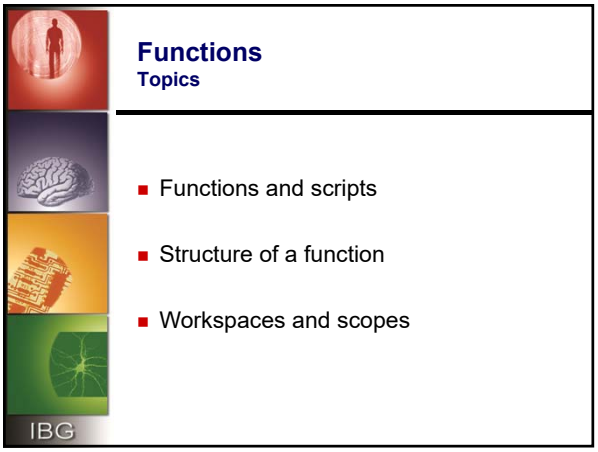
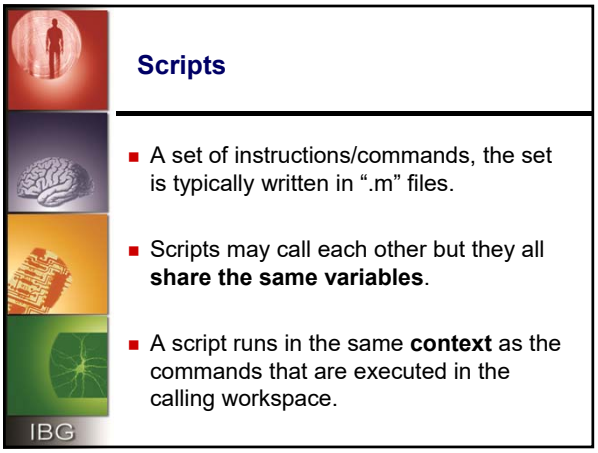

Introduction to programming
2017/18
Week 6: Functions
Izhar Bar-Gad
Room: 408 Phone: 7141 Email: izhar.bar-gad@biu.ac.il


Functions
Topics
Functions and scripts
Structure of a function
Workspaces and scopes
IBG


Scripts
A set of instructions/commands, the set is typically written in ".m" files.
Scripts may call each other but they all share the same variables.
A script runs in the same context as the commands that are executed in the calling workspace.
IBG

Script - example

- The file threePowers.m

```
disp([x x^2 x^3]);
x=x+1;
```

- Typing in the workspace

```
>> x=3; threePowers
   3    9    27

>> threePowers
   4   16   64
```

IBG

Scripts - problems

- Lack of separation between variables and environment of the various scripts.
- Requires explicit knowledge of the internal mechanism of all used scripts.
- Internal changes in one script alter the activity of other scripts.
- Impossible the divide a huge project into sub-projects.

IBG

Functions

- A set of instructions/commands that performs a specific operation encapsulated by an input/output interface.
- Input – A function *can* get variables as input – **input arguments**.
- Output – A function *can* return variables – **output arguments**.
- The input and output argument are the only way for the function to interact with the rest of the software.

There are of course exceptions that will be discussed in the final parts of the lecture.

IBG

Functions – Why?

- Logical separation of a big project into smaller potions.
- Simple ‘digestible’ code which is easier to maintain and debug.
- Separate a project to multiple programmers.
- Allows for portability of code.
- Avoid code duplication by allowing different programs or different parts within the same program to use the function.

Functions - syntax

- There are several types of functions, the most common type is defined and stored in a ‘.m’ file
- The file starts with the following line:

function [out1,out2, ..] = function_name(in1, in2, …)

- ‘function’ is a **reserved word**.
- out1,…: names of the output arguments (possibly none).
- function_name: the name of the function.
- in1,…: names of the input arguments (possibly none).

Function - Example

```
% The contents of the file stupidAppear.m

function result = stupidAppear(array,value)

if (any(array==value))
    disp(‘Found’);
    result = 1;
else
    disp(‘Not Found’);
    result = 0;
end
```

Calling a Function I

- Calling a function: By typing its name with the appropriate arguments.
- Syntax:
  [out1, out2, …] = function_name(in1, in2, …);
- Example:

```
isValue = stupidAppear(data,13);
```

The arguments can be values (not necessarily variables):

```
isValue = stupidAppear([1 24 14 13 24 13],13);
```

IBG

Calling a function II

- The names of input & output arguments may be different in the calling and called function.
- Example:

The function:

```
function y=doSomething(x)
```

May be called by:

```
a=doSomething(b);
```

IBG

Calling a function III

- A function may have no parameters
  - Example
    y = noParmFunc();
    or y = noParmFunc;
- The function is found according to a path of directories.

IBG

Function – 'Black Box'

A function is a 'black box' – when we call it, we only care **what** it does and how to **interface** with it, we don't mind **how** it does it.

Scopes

- A variable name and value has a meaning only within a **scope**.
- Different variables may have the same name in different scopes.
- A new scope is created with every call to a function and is destroyed every time you leave it.

Local variables

- Variables are generated and stored locally within the function they are removed from memory upon exit from the function.
- Input variables are a duplication of the variables used to call the function.

Local variables – example 1

Main script

```
x = 2;

disp('Before function');
disp(x);

myFunc(x)

disp('After function');
disp(x);
```

File – myFunc.m

```
function myFunc(x)

x = 3;

disp('In Function');
disp(x);
```

Local variables – example 2

Main script

```
x = 2;

myFunc(x)
disp('After first call');
disp(x);

myFunc(x)
disp('After second call');
disp(x);
```

File – myFunc.m

```
function myFunc(x)

x = x + 1;

disp('In Function');
disp(x);
```

Local variables – example 3

Main script

```
x=2;

x = myFunc(x)
disp('After first call');
disp(x);

x = myFunc(x)
disp('After second call');
disp(x);
```

File – myFunc.m

```
function a = myFunc(x)

a = x + 1;

disp('In Function');
disp(x);
```

Global variables

- Global variables are accessible by all functions and exist within the global scope.
- The command *global* is used to create global variables.
- **Example:** global firstVar secondVar;
- Do not use global variables !!!

Global variables – example

Main script

```
global y;
x = 2; y = x;

myFunc(x)
disp('After first call');
disp([x y]);

myFunc(x)
disp('After second call');
disp([x y]);
```

File – myFunc.m

```
function myFunc(x)

global y;
x = x + 1;
y = y + 1;

disp('In Function');
disp([x y]);
```

Function scopes

- A function has access to two basic scopes:
  - Its local scope
  - Global scope
- Typing in the command line also has these two types of scopes.
- A script shares its scope with its parent (or caller).

Persistency

- The *persistent* command defines variables that are local to the function in which they are declared yet their values are retained in memory between calls to the function.
- It is important to remember that the variable needs to be initiated during the first call.

Persistent variables – example

File – regularSquare.m

```
function  regularSquare()

if (~exist('y'))
    y=0;
else
    y=y+1;
end
disp(y^2);
```

File – persistentSquare.m

```
function persistentSquare()

persistent y;
if (~exist('y'))
    y=0;
else
    y=y+1;
end
disp(y^2);
```

Sub-function

- MATLAB enables calling a sub-function within the same file which cannot be called from functions outside the file.
- Example, file superFunc.m

```
function superFunc(x)
for i=1:1:100
    x=miniFunc(i+x);
end

function z=miniFunc(y)
disp(y);
z=y*2;
```

Functions in MATLAB

- The default installation of MATLAB contains many functions
  - Built-in functions
  - Toolboxes
- In addition functions are publicly available from many source for performing many procedures.
- It is usually better to use an external function rather than writing your own.
- A good source is:
  http://www.mathworks.com/matlabcentral/fileexchange/

Naming functions

- Name of the file and the function SHOULD ALWAYS be the same.
- In their simplistic form, every function is saved in a different file.
- There are multiple standards for names. My favorite is identical to the one for variables.
- Avoid **shadowing** by variables or other functions by prefixing with project name or your name.