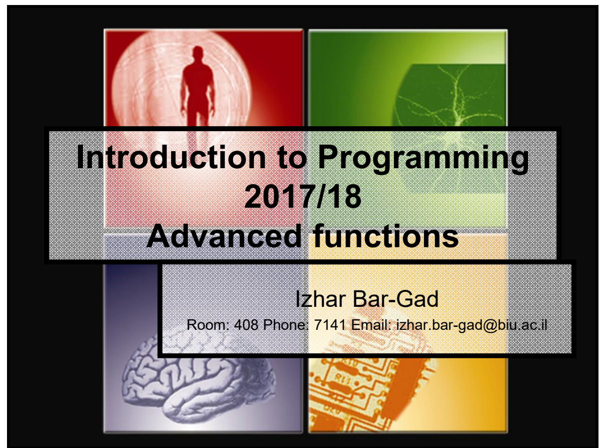
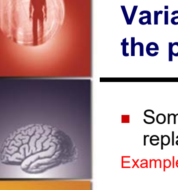


## Week 11 - Advanced functions

- Variable number of arguments
- Local function
- Functions as arguments
- Recursion



## Variable number of arguments – the problem

- Some parameters given to the function may be replaced by defaults:

Example: plot(y) == plot(y,'b') == plot(1:length(y),y,'b')

- Some functions perform the same operation on a large (non-fixed) number of parameters:

Example: save('izhar.mat','x','y','z',…)

- The problem is that functions, as we currently know them, have a fixed number of input and output arguments…

IBG

## Variable number of arguments – the solution

- Addressing the need for “non-fixed” number of input and output arguments is performed through the variables ***varargin*** & ***varargout***.
- Both variables are **cell arrays** and appear as the last arguments of the **input** & **output** arguments of the function respectively.
- Syntax:

function [var1,…,*varargout*] = myFunc(var1,…,*varargin*)

## VarArg example I

- Plot a circle of a given radius using a specified color and line width. However, they can both be omitted in favor of default values.

## VarArg example II

```
function myPlot(radius,varargin)

if (length(varargin)>0)
    circleColor = varargin{1};
else
    circleColor = ‘b’;
end
a =[0:.1:2*pi];
myHandle = plot(sin(a)*radius, cos(a)*radius, …
    circleColor);
if (length(varargin)>1)
    set(myHandle, ’lineWidth’, varargin{2});
end
```

## VarArg usage

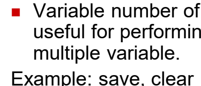
- Variable number of input arguments is very useful for accommodating default values.

Example: plot, stairs

- Variable number of input arguments is also useful for performing the same operation on multiple variable.

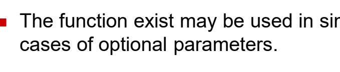
Example: save, clear

- Variable number of output arguments is typically less common as it requires special care of the **calling function**.



## VarArg – the simpler option

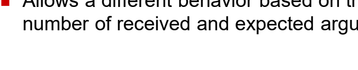
- The function exist may be used in simple cases of optional parameters.



Example:

```
function myFunc(nonOptVar, optVar)
…
if(~exist('optVar','var'))
   optVar=defaultValue;
end
…
```

## Number of arguments



- Number of arguments received - nargin
- Number of arguments expected out – nargout
- Allows a different behavior based on the number of received and expected arguments.
- Note: the number of arguments is the total number and not just those assigned to varargin or varargout.

## Local function I

- Local functions are any additional functions (to the primary function) within the same file.
- Local functions are only visible to the primary function or to other local functions in the same file.

## Local function II

- 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;
```

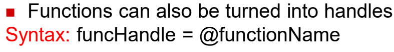
- Within the work environment:

>> superFunc(3) → OK
>> miniFunc(3) → Illegal

## Private functions

- Private functions reside in subdirectories with the special name **private**. They are visible only to M-file functions and M-file scripts that meet these conditions:
  - A function that calls a private function must be defined in an M-file that resides in the directory immediately above that private subdirectory.
  - A script that calls a private function must itself be called from an M-file function that has access to the private function according to the above rule.

## Function handles



- Functions can also be turned into handles:
Syntax: funcHandle = @functionName
- A function handle is a MATLAB value that provides a means of calling a function indirectly.
- Function handles may be used in calls to other functions (called *function functions*) or may be stored in data structures for later use.
- A function handle is one of the standard MATLAB data types.

IBG

## Function handles

- Function handles may be stored in a cell or structure arrays but not in standard arrays.

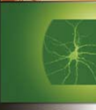
Example:
```
f.mySin = @sin;
```
Example:
```
f = {@cos @sin @log};
plot(f{1}([0:.1:2*pi]));
```
- The functions *func2str* & *str2func* are used to convert function handles to/from strings.

IBG

## Inline functions

- The *inline* function is used to construct an inline object.

Example:
```
>> f=inline('x.^y','x','y');
>> f(5,2)
   25
```



- The inline object may be used in the same manner as a function handle.

IBG

## Function functions I

- Functions may be passed as arguments using the function handle.
- Some system functions accept function names as strings in addition to handles.

Example:

*fzero* finds numerically zero crossing of a function

```
myFunc is a regular function
fzero('myFunc',5) or fzero(@myFunc,5)
or
myFunc = inline('(x-5)^2-4');
fzero(myFunc,5)
```

## Function functions II

- One of the common uses of function functions is for numerical analysis tools.
  - Find local minimum
    fminsearch(funcHandle, beginPoint)
    Example: fminsearch(@sin,2);
  - Numerically calculate the integral of a function
    quad(funcHandle, beginInt, endInt)
    Example: quad(inline('x.^2'),0,2)
- Other such functions
  fzero fminbnd fplot ode23 …

## Evaluation – problem & solution

- Problem
  - The MATLAB code cannot change while the program is running (static code).
  - Sometimes the code must be altered depending on user input or the results of a calculation.
- Solution
  - A new construct for dynamic code allows its generation during runtime.

## Evaluation I

- The function *eval* executes the string it receives.
- Simple example:

```
myVar = input('Variable name? ');
myStr = [myVar '=3'];
eval(myStr);
```

## Evaluation II

- Example:

```
for n = 4:8
    myStr = ['M' num2str(n) ' = magic(n)'];
    eval(myStr)
end
```

- *feval* may be used in the same way for function evaluation although direct calls may be performed through function handle.

## Recursion

- Recursion in computer programming defines a function in terms of itself.

**General structure**

```
function y=recfunc(x)
    …
    h=recfunc(g);
    …
```

## Recursion in scripts

- A script may call itself while maintaining the same variables.

```
% Contents of tmptmp.m
fprintf('Before %d\n', tmpVar);
tmpVar=tmpVar+1;
if (tmpVar<5)
   tmptmp;
end
fprintf('After %d\n', tmpVar);
```

What happens if we define tmpVar=1 and call tmptmp?

## Recursion in functions

- A function may call itself while generating a new set of variables.

```
% Contents of tmptmp.m
function tmptmp(tmpVar)
disp(['Before ' mat2str(tmpVar)]);
tmpVar=tmpVar+1;
if (tmpVar<5)
   tmptmp(tmpVar);
end
disp(['After ' mat2str(tmpVar)]);
```

What happens if we call tmptmp(1)?

## Recursion in functions

- A function may call itself while generating new variables for each call.

```
% Calculating the factorial of x
function y=factorial(x)
 if x == 0
   y = 1;
else
   y = factorial(x-1) * x;
end
```

- How can this be done non-recursively?

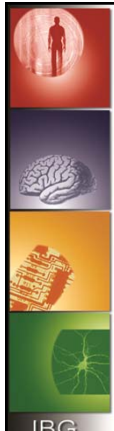
## Recursion and circularity

- Recursion is dangerous and may result in infinite calculations.

For example consider the dictionary term:
- **Recursion**
  See "Recursion".

- The key is the termination condition.

- The recursion may be indirect
  - Function *a* calls *b* which in turns calls *a.*




## Recursion – real life

- Many critical algorithms in computer science are based on recursion, such as searching for a value, sorting a list, etc.

- We will examine an example of finding a variable within an ordered array.

## Lion in the desert

- There is a lion loose in the desert, and it's our job to build a fence around it so it doesn't hurt anyone. We don't want to catch it any other way, since it's a very cranky lion. How should we do it?

- Answer: build a fence dividing the desert into two halves. The lion is now in one of the halves. Next, build a fence across the half containing the lion, constraining the lion to one quarter of the desert. Continue building fences across the half of the remaining desert containing the lion, until the lion is cornered.

## Find in an ordered array

- We have an array ***V*** that was pre-ordered in ascending order and would like to find out whether it contains an element ***e***.

```
function foundFlag = oFind(v,e)

c = ceil(length(v)/2);

if (length(v)==0)
   foundFlag=0;
elseif v(c) == e
   foundFlag = 1;
elseif v(c) < e
   foundFlag = oFind(v(c+1:end),e);
else
   foundFlag = oFind(v(1:c-1),e);
end
```