



# Lecture 7
# Programmer-Defined Functions in MATLAB

# Outline

- Why programmer-defined functions?
- Simple Custom Functions
- Script vs Function
- Modular Programming

# Why programmer-defined functions?

```
% Calculate x!/y!

fx=1;
for i=1:x
   fx=fx*i;
end

fy=1;
for i=1:y
   fy=fy*i;
end

Z=fx/fy;
disp(Z);
```

**Using functions**

```
% Calculate x!/y!
fx=fact(x);
fy=fact(y);
Z=fx/fy;
disp(Z);
```

- Less errors
- Better code organization
- More readable code
- Reuse of my functions
- Facilitate writing powerful/longer programs

# Functions Examples

**MATLAB's Built-in**

```
Y=sqrt(x);

Y=round(x);

Y=pow(x,3);

Y=rem(x,2);
```

**What is in red called:**
- **Function input(s)**
- **Input(s) to function**

**What is in black called:**
- **Function return(s)**
- **Function output(s)**

**Programmer-defined**

```
fx=fact(x);

rad= degree2rad(deg);

deg=rad2degree(rad);

d=dist(x1,y1,x2,y2);
```

# Simple Custom Functions

**Definition:** fact calculates the factorial of a number
**Input:** a number (x)
**Output:** factorial of x (fx)
**Function body:**

```
function fx=fact(x)
% calculates the factorial of a number
    fx=1;
    for i=1:x
        fx=fx*i;
    end
end
```

```
fx=1;
for i=1:x
    fx=fx*i;
 end
```

# Simple Custom Functions

```
function fx = fact(x)
% fact calculates the factorial of x
  fx=1;
  for i=1:x
      fx=fx*i;
   end
end
```

- Functions are contained in M-files
- The function name and file name should match
  - so the function **fact** should be in **fact.m**
- For help about the distance function use

  >> help fact
- Now in matlab I can use it the same way as the built-in function

```
>> f=fact(5)
   120
```

See MATLAB →→

# Simple Custom Functions

```
function rad= degree2rad(deg)
%degree2rad converts the input degree  into radians
      rad=deg*pi/180;
end
```

```
function deg= rad2degree(rad)
%rad2degree converts the input radians into degrees
    deg=rad*180/pi;
end
```

```
function dist = distance(x1, y1, x2, y2)
%distance calculates the distance between two points (x1,y1) and (x2, y2)
    dist = sqrt((x2-x1)^2+(y2-y1)^2);
end
```

# Error Check Inputs

```
function x = correctInput(msg,low,high)
% correctInput prompt the user using msg for an input and
% error check to  make sure it is >= low and <=high
x=input(msg);
while x<low || x> high
   fprintf('Error! ');
   x=input(msg);
end
end
```

**correctInput.m**

```
>> x=correctInput('enter num:',0,10);
enter num:-5
Error! enter num:11
Error! enter num:4
>>
>> x
x =
    4
```

← **Command Window**

# Custom Function with no return

```
function drawLine(n,ch)
% draw line of length n using the character ch.
for i=1 to n
    fprintf('%s',ch);
end
fprintf('\n');
```

**drawLine.m**

```
>> drawLine (5,'*');
*****

>> drawLine (10,'X');
XXXXXXXXXX

>> drawLine(5,'-x-');
-x--x--x--x--x-
```

← **Command Window**

# Custom Function with no input

```
function x=getPositiveNumber
% read positive number from the user
x=input('Enter positive number:');
while x<0
   fprintf('Error! ');
   x=input('Enter positive number:');
end
end
```

**getPositiveNumber.m**

```
>> x=getPositiveNumber
Enter positive number:-3
Error! Enter positive number:-5
Error! Enter positive number:5

x =

    5
```

← **Command Window**

# Custom Function with neither input nor return

```
function drawFixedLine
% draw a line of fixed length 30 using the shape -x-.
for i=1:30
    fprintf('-x-');
end
fprintf('\n');
end
```

**drawFixedLine.m**

**Command Window**

```
>> drawFixedLine
-x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x-

>> drawFixedLine
-x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x--x-
```

# Custom Function with more returns

```
function [large, small] = sort2(x,y)
%sort2 compares the two inputs and return them sorted, larger first.
if x>=y
   large=x;
   small=y;
else
   large=y;
   small=x;
end
end
```

**sort2.m**

```
>> [x,w]=sort2(5,10)
x =
   10
w =
    5
```

```
>> [x,w]=sort2(10,5)
x =
   10
w =
    5
```

# Custom (User-Defined) Functions

```
function [out_arg1, out_arg2, …]
    = fname(in_arg1, in_arg2, …)
```

- The word `function` is a keyword.
- `[out_arg1, out_arg2, …]` is the output argument list
- `fname` is the name of the function
- `(in_arg1, in_arg2, …)` is the input argument list
  - `in_arg1`, `in_arg2`, etc. are called “dummy arguments” because they are filled will values when the function is called

# Local Workspace

**Script**

```
w=input('enter a number:');
y=fact(w);
disp(y);
```

Command Window Workspace contains w and y.
So fx, x, and i are not known here

**Function**

```
function fx = fact(x)
%fact calculates the factorial of x
fx=1;
for i=1:x
   fx=fx*i;
end
end
```

Function fact Workspace contains fx, x, and i

When a function reaches the end of execution (and returns the output argument), the function space— local space—is deleted.

# Script Vs. Function

**Script**

- A script is executed line-byline just as if you are typing it into the Command Window
- The value of a variable in a script is stored in the Command Window Workspace

**Function**

- A function has its own private (local) function workspace that does not interact with the workspace of other functions or the Command Window workspace
- Variables are not shared between workspaces even if they have the same name

# Modular Programming

Write program that takes the student grade (out of 100) in Math, Science, and English and prints ‘Excellent’ if greater than or equal 90%, ‘Very Good’ if greater than or equal 80% and smaller than 90%, ‘Good’ if greater than or equal 70 and smaller than 80%, ‘Fair’ if greater than or equal 60% and smaller than 70%,’Fail’ if lower than 60%.

**Sample Input/Output:**

Enter Math grade from 0 to 100:94

Enter Science grade from 0 to 100:87

Enter English grade from 0 to 100:77

You got Excellent in Math

You got Very Good in Science

You got Good in English

# Modular Programming

Write program that takes the student grade in Math, Science, and English

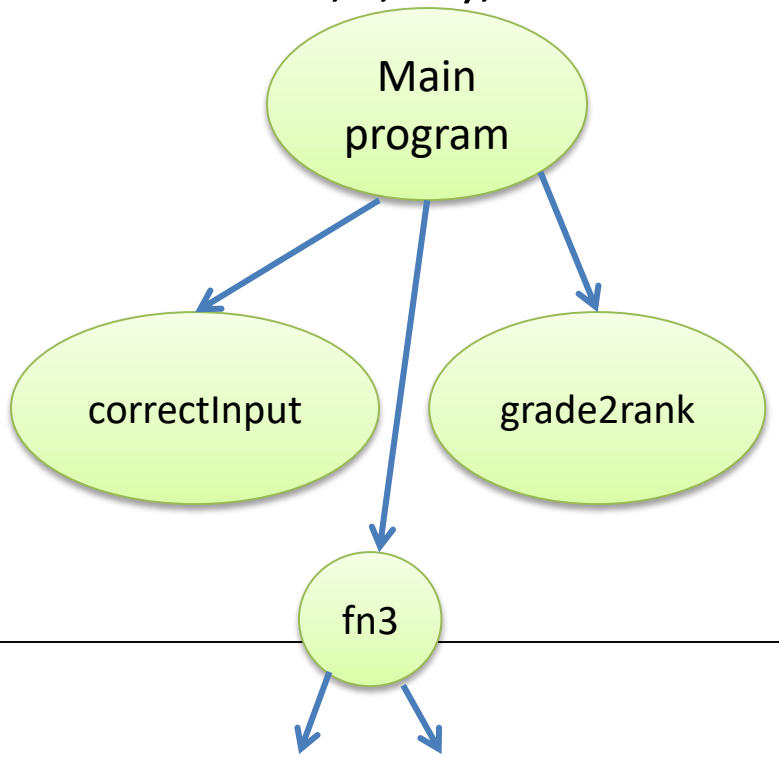
```
m=correctInput('Enter math grade from 0 to 100:',0,100);
s=correctInput('Enter science grade from 0 to 150:',0,150);
e=correctInput('Enter English grade from 0 to 50:',0,50);
```

and prints ‘Excellent’ if greater than or equal 90%, ‘Very Good’ if greater than or equal 80% and smaller than 90%, ‘Good’ if greater than or equal 70 and smaller than 80%, ‘Fair’ if greater than or equal 60% and smaller than 70%,’Fail’ if lower than 60%.

```
mr=grade2rank(m);        % A function to convert grade numeric to alphanumeric
sr=grade2rank(s);
er=grade2rank(e);

fprintf('You got %s in math\n',mr);
fprintf('You got %s in science\n',sr);
fprintf('You got %s in english\n',er);
```

# Use Functions for Clean and Organized Code (Modular Programs)

```
m=correctInput('Enter math grade from 0 to 100:',0,100);
s=correctInput('Enter science grade from 0 to 150:',0,150);
e=correctInput('Enter english grade from 0 to 50:',0,50);

mr=grade2rank(m);
sr=grade2rank(s);
er=grade2rank(e);

fprintf('You got %s in math\n',mr);
fprintf('You got %s in science\n',sr);
fprintf('You got %s in english\n',er);
```



# Function: grade2rank

```
function r=grade2rank(x)
% calculate the corresponding rank of the grade x

if x>=90
   r='Ecellent';
elseif x>=80
   r='Very Good';
elseif x>=70
   r='Good';
elseif x>=60
   r='Fair';
else
   r='Fail';
end
end
```



# Thank You

Course Site:
http://scholar.cu.edu.eg/?q=eldeib/classes/genn004-computers-engineers

**Computers for Engineers – GENN004**