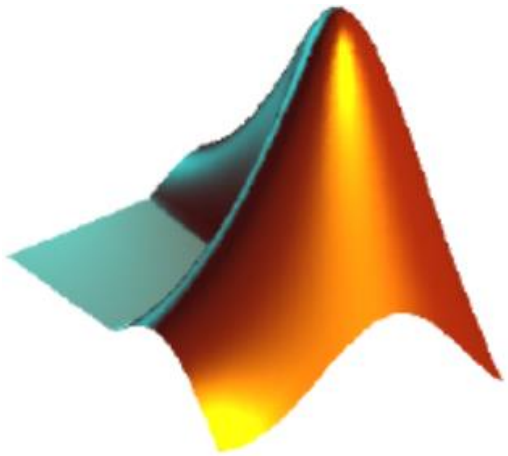
Matlab Programming
Scripts & Functions
(3)



M Files - Scripts

MATLAB allows writing two kinds of program files:

1. **Scripts:**
   - program files with .m extension.
   - Files with series of commands,(to execute together).
   - Scripts do not accept inputs and do not return any outputs.
   - They operate on data in the workspace.

Example:

```
a = 5; b = 7;
c = a + b
d = c + sin(b)
e = 5 * d
f = exp(-d)
```

Script Files Creation

- To **create** a **script file**, just type **edit** command and the filename .m

```
>> edit <filename>
```

- The created **file** will be placed in Matlab **current folder.**

Example:

```
>>mkdir progs % create directory progs under default folder (Optional)

>>chdir progs % changing the current directory to progs  (Optional)

>>edit prog1.m % creating an m file named prog1.m
```

Running a script

- Create the script ‘prog1.m’ and save it.

```
a = 5; b = 7;
c = a + b
d = c + sin(b)
e = 5 * d
f = exp(-d)
```

- To run a script, just type the script file name in the prompt command.

```
>> prog1
c =    12
d =    12.6570
e =   63.2849
f =    3.1852e-06
```

M Files - Functions

MATLAB allows writing two kinds of program files:

2. **Functions:**
   - program files with .m extension.
   - The function name must have the same name of the function file.
   - Functions can accept inputs and return outputs.
   - Internal variables are local to the function.

Example:

```
function max = mymax(n1, n2)
%This function calculates the maximum of two numbers
max = n1;
if(n2 > max)
max = n2;
end
```

Functions Creation

To **create** a **function**,

1- The function name must have the same name of the function file.

```
>> chdir progs                %optional

>> edit <functionname>
```

2- In the functionname.m file, type your function**.**

- The function syntax is:

```
function [out1,out2, ..., outN] = myfun(in1,in2,in3, ..., inN)
```

Function Example

- Write a **function** named **mymax** that takes as its **input five numbers**.

- It **returns** the **maximum** of the five numbers.

Solution:

1- Open a (.m) file and name it mymax.

```
>> chdir progs
>> edit  mymax.m
```

Function Example: Solution

2- Write the function in the mymax.m file as follows:

```
function max = mymax(n1, n2, n3, n4, n5)
%This function calculates the maximum of the five numbers given as input
max = n1;
if(n2 > max)
   max = n2;
end
if(n3 > max)
   max = n3;
end
if(n4 > max)
   max = n4;
end
if(n5 > max)
   max = n5;
end
```

Calling a Function

- The **comment lines** that come right after the function statement provide the **help** text.
- These lines are printed when you type:

```
>> help mymax
This function calculates the maximum of the five numbers given as input
```

- To **call** a **function**, type its name and pass its input parameters

```
>>mymax(34, 78, 89, 23, 11)
    ans = 89
```

Anonymous Functions

- An **anonymous function is** like an **inline function**, in traditional programming.
- It is defined within a **single MATLAB statement**.
- It consists of a **single MATLAB expression** and **any number of input** and **output arguments**.
- You can **define** an **anonymous function** right at the MATLAB **command line or** within a **function or script**.

Anonymous Functions: Example

- Syntax:

```
f = @(arglist)expression
```

- Example:

  Write an **anonymous function** named **power**, which will take **two numbers** as **input** and return first number raised to the power of the second number.

```
power = @(x, n) x.^n;
result1 = power(7, 3)
result2 = power(49, 0.5)
result3 = power(10, -10)
result4 = power (4.5, 1.5)
```

Anonymous Functions Example: Solution

- Create a script file

```
>> chdir progs
>>edit pow.m
```

- Write the following code in it

```
power = @(x, n) x.^n;
result1 = power(7, 3)
result2 = power(49, 0.5)
result3 = power(10, -10)
```

- Run the file

```
>> pow
    result1 = 343
    result2 =  7
    result3 = 1.0000e-10
```

Primary and Sub-Functions

- Any **function other than an anonymous function must** be **defined** within a **file**.
- Each **function file contains** a **required primary function** that **appears first and** any **number of optional sub-functions** that comes **after the primary function** and used by it.
- **Primary functions** can be **called** from **outside of the file that defines them**, either **from command line or** from **other functions**.
- **Sub-functions cannot** be **called** from **command line or** other **functions outside the function file**.
- **Sub-functions** are **visible only** to the **primary function and other sub-functions within the function file** that defines them.

Primary and Sub-Functions: Example

Example:

- Write a **function** named **quadratic** that would **calculate** the **roots of a quadratic equation**.

- The function would take **three inputs**, the quadratic co-efficient, the linear co-efficient and the constant term.

- It would **return** the **roots**.

- The function file **quadratic.m** will **contain** the **primary function** ***quadratic*** *and the* ***sub-function disc****, which* ***calculates*** *the* ***discriminant****.*

Primary and Sub-Functions Example: Solution

1- Open a (.m) file and name it quadratic.

```
>>chdir progs;   edit quadratic.m
```

2- Write the function in the quadratic.m file as follows:

```
function [x1,x2] = quadratic(a,b,c)
d = disc(a,b,c);        %calling the sub-function 'disc'
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end                     % end of quadratic

function dis = disc(a,b,c)
%a sub function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);
end % end of sub-function
```

3- Calling the function:

```
>> quadratic(2,4,-4)
```

Nested Functions

- **Nested functions: define functions within** the **body** of **another function.**

- **Nested functions** are defined within the scope of another function and they share **access** to the **containing function's workspace**.

- A nested function follows the below syntax:

```
function x = A(p1, p2)
...
B(p2)
function y = B(p3)
...
end
...
end
```

Nested Function: Example

Rewrite the **function *quadratic***, *from previous example, however, this time the* ***disc*** *function will be a* ***nested function***.

1- Open a (.m) file and name it quadratic2.

```
>>chdir progs;   edit quadratic2.m
```

2- Write the function in the quadratic.m file as follows:

```
function [x1,x2] = quadratic2(a,b,c)

function disc     % nested function
d = sqrt(b^2 - 4*a*c);
end               % end of function disc

disc;
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end               % end of function quadratic2
```

3- Calling the function:

```
>> quadratic2(2,4,-4)
    ans = 0.7321
```

Private Functions

- A **private function** is a **primary function** that is **visible** only to a **limited group of other functions.**

- If you do **not** want to **expose** the **implementation** of a **function(s**), you can **create** them as **private** functions.

- **Private functions** reside in **subfolders** with the **special name 'private'.**

- **Private functions** are **visible** only to **functions in** the **parent folder**.

Example:

- **Rewrite** the ***quadratic function***. *This time, however, the* ***disc*** *function calculating the discriminant, will be a* ***private function***.

Private Function: Example

1- **Create** a **subfolder** named **private** in working directory. Store the following function file ***disc.m*** *in it:*

```
function dis = disc(a,b,c)
%function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);
end % end of sub-function
```

2- **Create** a **function quadratic3.m** in your working directory and type the following code in it:

```
function [x1,x2] = quadratic3(a,b,c)
d = disc(a,b,c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end % end of quadratic3
```

3- Calling the function:

>> quadratic3(2,4,-4)

ans = 0.7321

Global Variables

- **Global variables** can be **shared** by **more than one function**.
- For this, you need to **declare** the **variable as global in all the functions**.
- To **access that variable** from the base **workspace**, then **declare** the **variable** at the **command line**.
- The **global declaration must occur before** the **variable** is actually **used in** a **function**.
- It is a good practice to use **capital letters** for the **names** of **global variables** to **distinguish them** from **other variables**.

Example:

- **Create a function average.m** that calculates the **average** of the **input numbers**.

Global Variables Example: Solution

1- Open a (.m) file and name it average.

```
>>chdir progs;   edit average.m
```

2- Write the function in the **average.m** file as follows:

```
function avg = average(nums)
global TOTAL
avg = sum(nums)/TOTAL;
end
```

3- Open a script file and name it **sp.m**. In the prompt type:

```
>> chdir progs
>>edit sp.m
```

4- Write the following code in it:

```
global TOTAL;
TOTAL = 10;
n = [34, 45, 25, 45, 33, 19, 40, 34, 38, 42];
av = average(n)
```

5- call the srcipt file from the prompt by typing:

```
>>sp
```

av= 35.5000 142