Computer Programming: User-Defined Functions

Asst. Prof. Dr. Yalçın İşler
Izmir Katip Celebi University

Outline

- Introduction
- Creating a Function File
- Structure
- Local and Global Variables
- Saving and Using Function Files
- Comparing to Script Files
- Anonymous and Inline Functions
- Functions on Functions
- Subfunctions
- Nested Functions

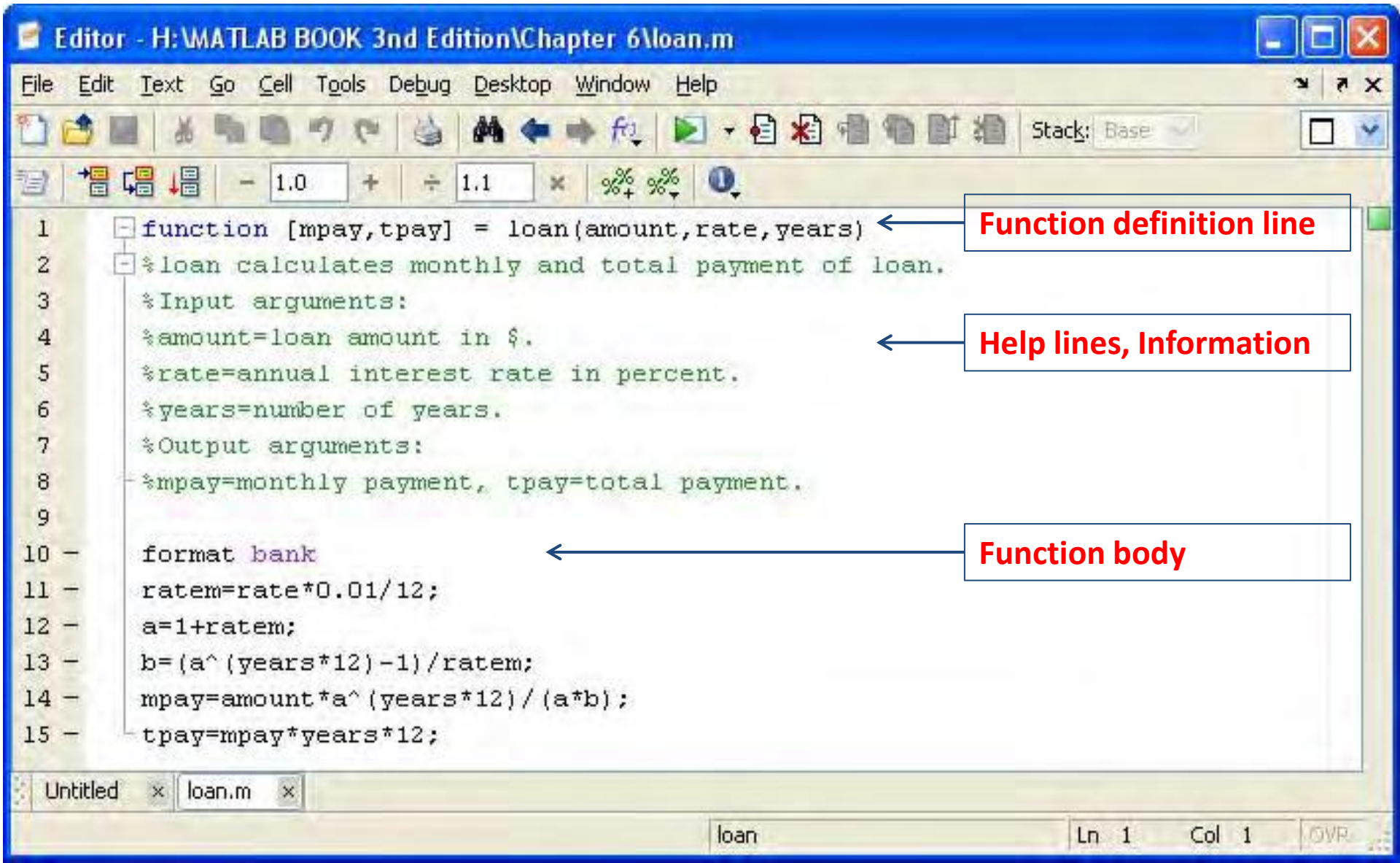
Introduction

- A simple function in mathematics, y, associates a unique number to each value of *x.*
- For example: sin(x), cos(x), sqrt(x), exp(x), rand(N), etc.
- When a function expression is simple and needs to be calculated only once, it can be typed as part of the program. However, when a function needs to be evaluated many times for different values of arguments it is convenient to create a “user-defined” function. Once a user-defined function is created (saved) it can be used just like the built-in functions.

Creating a Function File

- Function files are created and edited, like script files, in the Editor/Debugger Window.

Structure


Editor - H:\MATLAB BOOK 3nd Edition\Chapter 6\loan.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1
1 function [mpay,tpay] = loan(amount,rate,years)
2 %loan calculates monthly and total payment of loan.
3 %Input arguments:
4 %amount=loan amount in $.
5 %rate=annual interest rate in percent.
6 %years=number of years.
7 %Output arguments:
8 %mpay=monthly payment, tpay=total payment.
9
10 format bank
11 ratem=rate*0.01/12;
12 a=1+ratem;
13 b=(a^(years*12)-1)/ratem;
14 mpay=amount*a^(years*12)/(a*b);
15 tpay=mpay*years*12;
Function definition line
Help lines, Information
Function body
Untitled
loan.m
loan
Ln 1
Col 1
OVR

Structure Of a Typical Function File

- The first executable line in a function file **must be the function definition line.** Otherwise the file is considered a script file.

- The input and output arguments are used to transfer data into and out of the function. The input arguments are listed inside parentheses following the function name. Usually, there is at least one input argument, although it is possible to have a function that has no input arguments. If there are more than one, the input arguments are separated with commas. In the example shown, there are three input arguments: (amount,rate,years).
- The output arguments, which are listed inside brackets on the left side of the assignment operator in the function definition line, transfer the output from the function file. Function files can have none, one, or several output arguments. If there are more than one, the output arguments are separated with commas. If there is only one output argument it can be typed without brackets. In the example, there are two output arguments: [mpay,tpay].

- The help text lines are comment lines (lines that begin with the percent % sign) following the function definition line. They are optional, but frequently used to provide information about the function.

- The function body contains the computer program (code) that actually performs the computations.

help

- Shows the information lines given in the function. The comment lines that are typed between the function definition line and the first non-comment line are displayed when the user types help function_name in the Command Window. For example,

```
>> help loan
loan calculates monthly and total payment of loan.
Input arguments:
amount=loan amount in $.
rate=annual interest rate in percent.
years=number of years.
Output arguments:
mpay=monthly payment, tpay=total payment.
```

Local and Global Variables

- All the variables in a function file are local (the input and output arguments and any variables that are assigned values within the function file). This means that the variables are defined and recognized only inside the function file. When a function file is executed, MATLAB uses an area of memory that is separate from the workspace (the memory space of the Command Window and the script files).
- The variable has to be declared global in every function file that the user wants it to be recognized in. The variable is then common only to these files.

Global Variables

- Several variables can be declared global by listing them, separated with spaces, in the global command.
- The global command must appear before the variable is used. It is recommended to enter the global command at the top of the file.
- The global command has to be entered in the Command Window, or/and in a script file, for the variable to be recognized in the workspace.
- The variable value can be assigned, or reassigned, a value in any of the locations it is declared common.
- It is recommended to use long descriptive names (or use all capital letters) for global variables in order to distinguish between them and regular variables.

Saving Function Files

- A function file must be saved before it can be used. It is necessary that the file is saved with a name that is identical to the function name in the function definition line.
- A user-defined function is used in the same way as a built-in function. The function can be called from the Command Window, from a script file, or from another function.

Using Function Files

```
>> [month total]=loan(25000,7.5,4)

month =
         600.72
total =
       28834.47
```

First argument is loan amount, second is interest rate, and third is number of years.

```
>> a=70000;  b=6.5;
>> [x y]=loan(a,b,30)

x =
         440.06
y =
      158423.02
```

Define variables `a` and `b`.

Use `a`, `b`, and the number 30 for input arguments and `x` (monthly pay) and `y` (total pay) for output arguments.

Comparing to Script Files

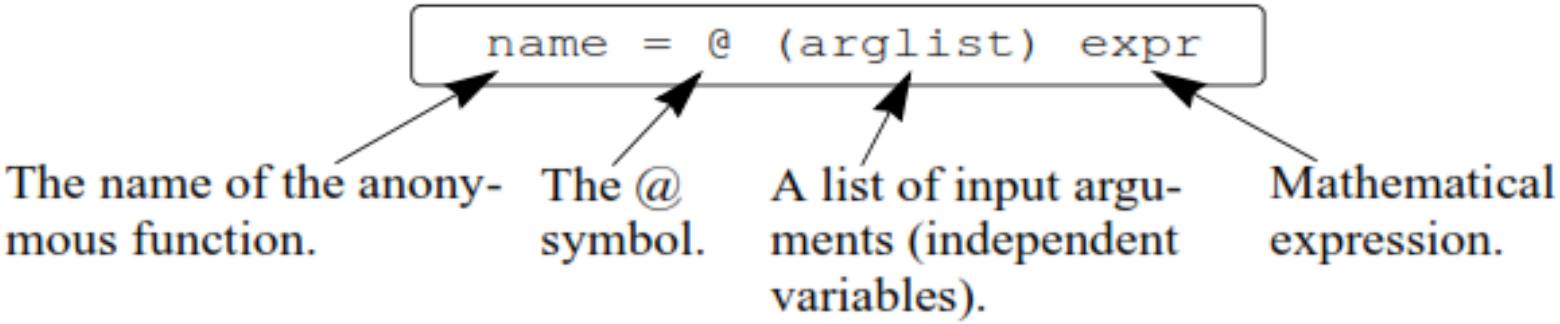
- Both script and function files are saved with the extension .m (that is why they are sometimes called M-files).
- The first line in a function file is the function definition line.
- The variables in a function file are local. The variables in a script file are recognized in the Command Window, and vice versa.
- Both contain a sequence of MATLAB commands (statements).
- Function files can accept data through input arguments and can return data through output arguments.
- When a function file is saved, the name of the file should be the same as the name of the function.

Anonymous and Inline Functions (1)

- An anonymous function is a user-defined function that is defined and written within the computer code (not in a separate function file) and is then used in the code.
- They replace inline functions that have been used for the same purpose in previous versions of MATLAB.

Anonymous Function Examples (1)

An anonymous function is created by typing the following command:



A simple example is: `cube = @ (x) x^3`, which calculates the cubic power of the input argument.

An example of an anonymous function that has two independent variables is:

```
circle = @ (x,y) 16*x^2+9*y^2
```

The expression can include predefined variables that are already defined when the anonymous function is defined. For example, if three variables `a`, `b`, and `c` are defined (they have assigned numerical values), then they can be used in the expression of the anonymous function: `parabola = @ (x) a*x^2+b*x+c`.

Anonymous Function Examples (2)

The function: $f(x) = \dfrac{e^{x^2}}{\sqrt{x^2+5}}$ can be defined (in the Command Window) as an inline function for x as a scalar by:

```
>> FA = @ (x) exp(x^2)/sqrt(x^2+5)
FA =
    @(x)exp(x^2)/sqrt(x^2+5)
```

If a semicolon is not typed at the end, MATLAB responds by displaying the function. The function can then be used for different values of x, as shown below.

```
>> FA(2)
ans =
   18.1994
>> z = FA(3)
z =
  2.1656e+003
```

Anonymous Function Examples (3)

If x is expected to be an array, and the function calculated for each element, then the function must be modified for element-by-element calculations.:

```
>> FA = @ (x) exp(x.^2)./sqrt(x.^2+5)
FA =
    @(x)exp(x.^2)./sqrt(x.^2+5)
>> FA([1 0.5 2])
ans =
    1.1097    0.5604   18.1994
```

Using a vector as input argument.

Anonymous and Inline Functions (2)

Similar to anonymous function, an inline function is a simple user-defined function that is defined without creating a separate function file (M-file). As already mentioned, anonymous functions replace inline functions used in earlier versions of MATLAB. Inline functions are created with the `inline` command according to the following format:

```
name = inline('math expression typed as a string')
```

A simple example is: `cube = inline('x^3')`, which calculates the cubic power of the input argument.

- cannot accept preassigned variables.
- can be used as an argument of other functions.

Inline Function Examples (1)

For example, the function: $f(x) = \frac{e^{x^2}}{\sqrt{x^2+5}}$ can be defined as an inline function for x by:

```
>> FA=inline('exp(x.^2)./sqrt(x.^2+5)')
FA =
     Inline function:
     FA(x) = exp(x.^2)./sqrt(x.^2+5)
>> FA(2)
ans =
   18.1994
>> FA([1 0.5 2])
ans =
    1.1097    0.5604   18.1994
```

Expression written with element-by-element operations.

Using a scalar as the argument.

Using a vector as the argument.

Inline Function Examples (2)

$f(x, y) = 2x^2 - 4xy + y^2$ can be defined as an inline function by:

```
>> HA=inline('2*x^2-4*x*y+y^2')
HA =
     Inline function:
     HA(x,y) = 2*x^2-4*x*y+y^2
```

Once defined, the function can be used with any values of x and y. For example, HA(2,3) gives:

```
>> HA(2,3)
ans =
    -7
```

Functions on Functions

- There are many situations where a function (Function *A) works on (uses) another* function (Function *B). This means that when Function A is executed it has to be* provided with Function *B.* For example, MATLAB has a built-in function called fzero (Function *A) that finds the zero of a math function (Function B), i.e. f(x), the value of x where f(x)=0.*
- *There are two methods:*
  - ***Using Function Handles for Passing a Function into a Function***
  - ***Using a Function Name for Passing a Function into a Function***

a function handle is created by typing the symbol @ in front of the function name.

Parsing Functions by Handle

A name for the function that is passed in.

```
function xyout=funplot(Fun,a,b)
% funplot makes a plot of the function Fun which is passed in
% when funplot is called in the domain [a, b].
% Input arguments are:
% Fun:  Function handle of the function to be plotted.
% a:  The first point of the domain.
% b:  The last point of the domain.
% Output argument is:
% xyout: The values of x and y at x=a, x=(a+b)/2, and x=b
% listed in A 3 by 2 matrix.

x=linspace(a,b,100);
y=Fun(x);
xyout(1,1)=a; xyout(2,1)=(a+b)/2; xyout(3,1)=b;
xyout(1,2)=y(1);
xyout(2,2)=Fun((a+b)/2);
xyout(3,2)=y(100);
plot(x,y)
xlabel('x'), ylabel('y')
```

Using the imported function to calculate $f(x)$ at 100 points.

Using the imported function to calculate $f(x)$ at the midpoint.

As an example, the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ over the domain $[0.5, 4]$ is passed into the user-defined function `funplot`. This is done in two ways: first, by writing a user-defined function for $f(x)$, and then by writing $f(x)$ as an anonymous function.

To use an anonymous function, the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ first has to be written as an anonymous function, and then passed into the user-defined function `funplot`.

```
>> FdemoAnony=@(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3
FdemoAnony =
    @(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3
```

Create an anonymous function for $f(x)$.

```
>> ydemo=funplot(@Fdemo,0.5,4)
ydemo =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

Enter a handle of the user-defined function `Fdemo`.

Parsing Functions by Name

The `feval` (function evaluate) command evaluates the value of a function for a given value (or values) of the function's argument (or arguments). The format of the command is:

```
variable = feval('function name', argument value)
```

```
>> ydemoS=funplotS('Fdemo',0.5,4)
ydemoS =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

The name of the imported function is typed as a string.

```
function xyout=funplotS(Fun,a,b)
% funplotS makes a plot of the function Fun which is passed in
% when funplotS is called in the domain [a, b].
% Input arguments are:
% Fun: The function to be plotted. Its name is entered as
string expression.
% a:  The first point of the domain.
% b:  The last point of the domain.
% Output argument is:
% xyout: The values of x and y at x=a, x=(a+b)/2, and x=b
% listed in A 3 by 2 matrix.

x=linspace(a,b,100);
y=feval(Fun,x);
xyout(1,1)=a; xyout(2,1)=(a+b)/2; xyout(3,1)=b;
xyout(1,2)=y(1);
xyout(2,2)=feval(Fun,(a+b)/2);

xyout(3,2)=y(100);
plot(x,y)
xlabel('x'), ylabel('y')
```

A name for the function that is passed in.

Using the imported function to calculate $f(x)$ at 100 points.

Using the imported function to calculate $f(x)$ at the midpoint.

Subfunctions

- A function file can contain more than one user-defined function. The functions are typed one after the other. Each function begins with a function definition line. The first function is called the primary function and the rest of the functions are called subfunctions. The subfunctions can be typed in any order. The name of the function file that is saved should correspond to the name of the primary function.

Nested Functions

- A nested function is a user-defined function that is written inside another user-defined function.
- The portion in the code that corresponds to the nested function starts with a function definition line and ends with an end statement.
- An end statement must be also entered at the end of the function that contains the nested function. (Normally, a user-defined function does not require a terminating end statement. However, an end statement is required if the function contains one or more nested functions.)
- Nested functions can also contain nested functions.

Nested Functions (cont'd)

```
function y=A(a1,a2)
.......
    function z=B(b1,b2)
    .......
    end
.......
end
```

```
function y=A(a1,a2)
.......
    function z=B(b1,b2)
    .......
    end
.......
    function w=C(c1,c2)
    .......
    end
.......
end
```

Laboratory Session

Do both examples and sample applications in Chapter 6 of the textbook.

Homework #8

Not later than the next week:

Solve problems 3, 8, 9, 14, and 19 from the Chapter 6 of the textbook using Matlab.