SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 5-Lecture 17

BTech (CSBS) -Semester VII

16 Septemer 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

Introduction to programming

1) Introduction,
2) M-File Scripts,
3) script side-effects,
4) M-File functions,
5) anatomy of a M- File function,
6) input and output arguments,
7) input to a script file,
8) output commands.

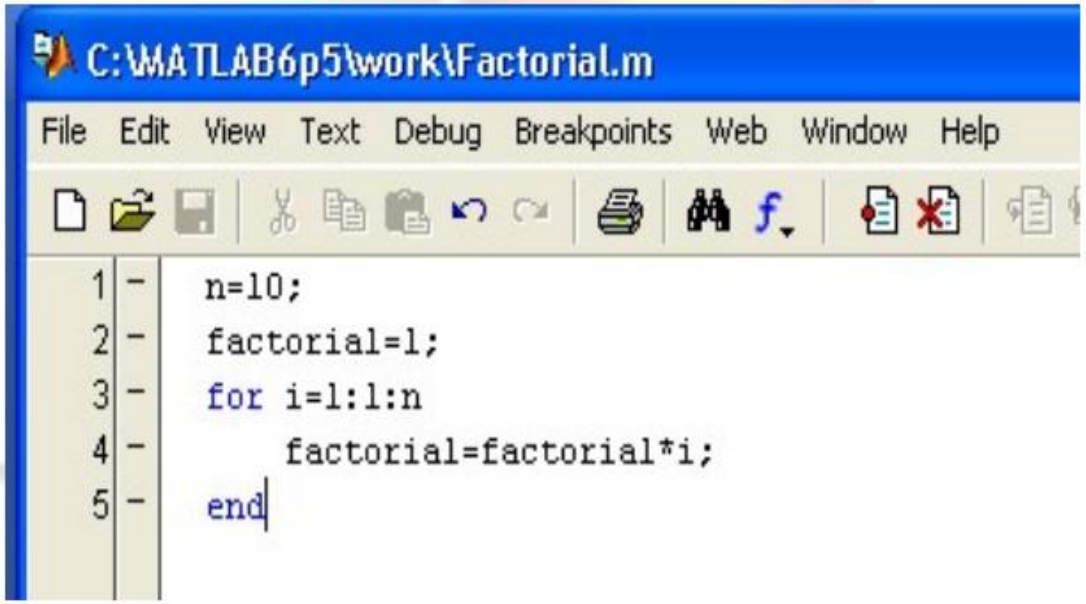
Kinds of M files

Script M-Files	Function M-Files
Do not accept input Do not accept input arguments or return arguments or return output arguments output arguments	Can Can accept accept input input arguments and return arguments and return output arguments output arguments
Operate on data in the workspace	Internal variables are local to the function by default
Useful for automating a series of steps you need to perform many times	Useful for extending the MATLAB language for your application

Example

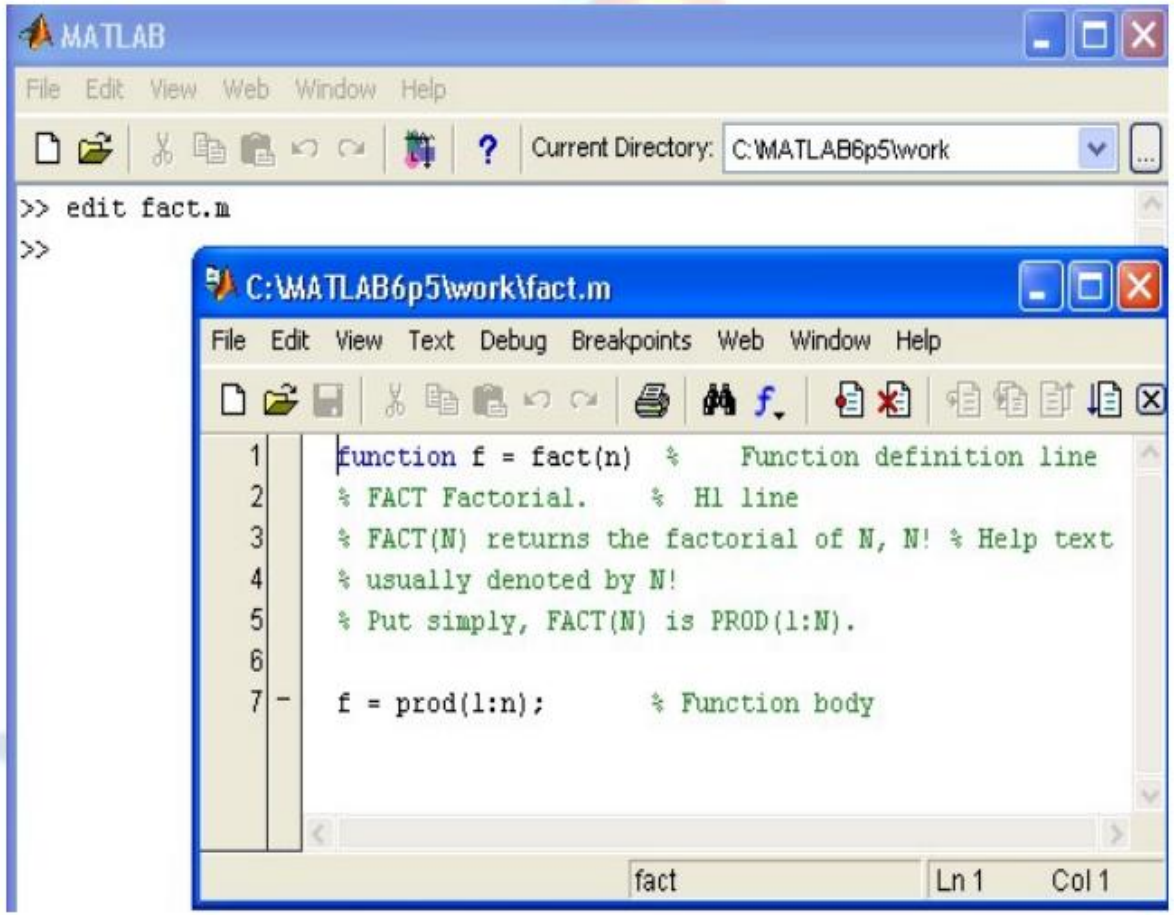
Factorial.m

```
n=10;
factorial=1;
for i=1:1:n
   factorial=factorial*i;
end
```



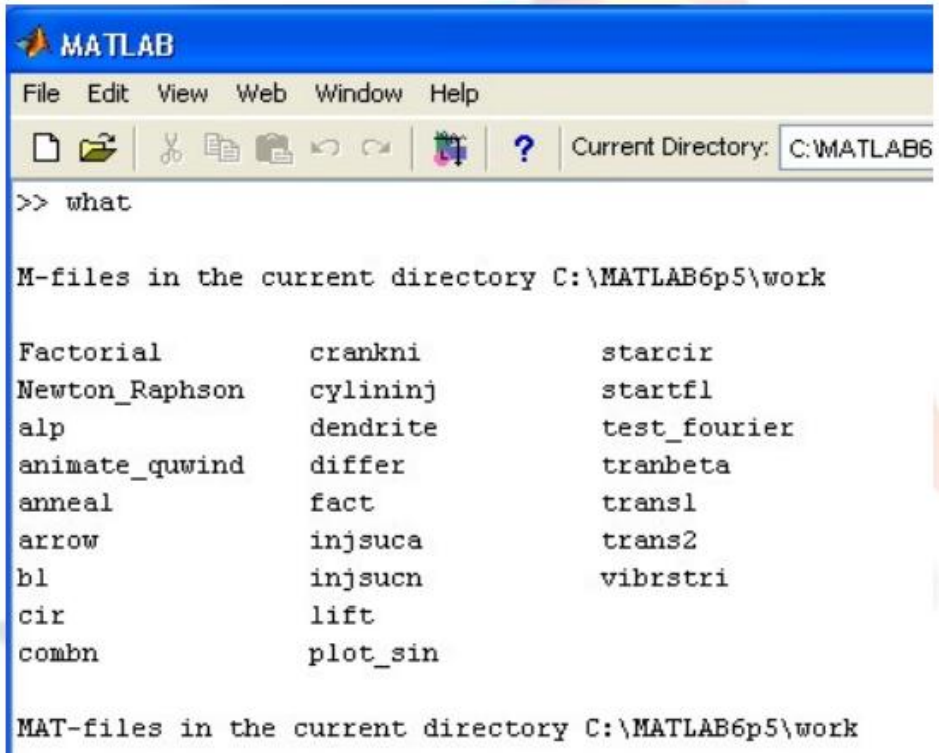
Accessing Text Editors

>> edit fact.m



Listing files

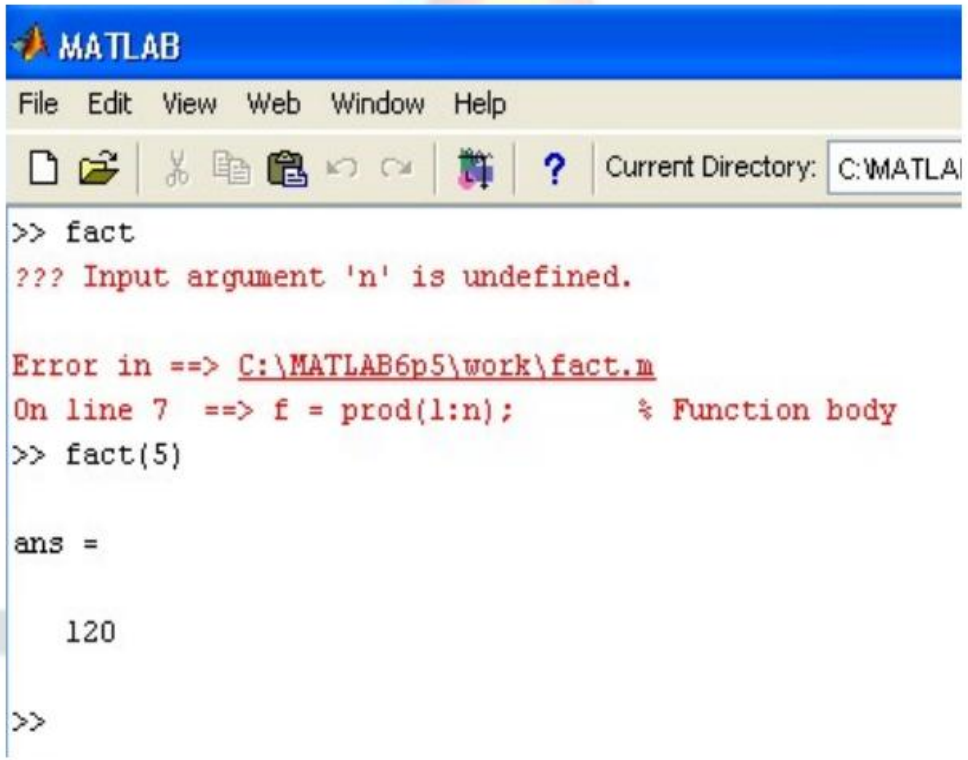
>> what (List the names of the files in your current directory)



Calling function

Call the fact function

>> fact(5)



Some m-File Functions

Function	Description
run	Run script that is not on current path
type filename	lists the contents of the file given a full pathname
Edit fun	opens the file fun.m in a text editor
mfilename	Name of currently running M-file
namelengthmax	Return maximum identifier length
echo	Echoes the script file contents as they are executed

Some m-File Functions

Function	Description
input	Request user input
Disp (variablename)	Displays results without printing variable names
beep	Makes computer beep
eval	Interpret strings containing MATLAB expressions
feval	Evaluate function

Some m-File Functions

Function	Description
pause	Pauses and waits until user presses any keyboard key.
pause (n)	Pause (n) pauses for n seconds and then continues
waitforbuttonpress	Pauses until user presses mouse button or any keyboard key
keyboard	Temporarily gives control to keyboard. ➢The keyboard mode is terminated by executing the command RETURN ➢DBQUIT can also be used to get out of keyboard mode but in this case the invoking M-file is terminated.

Input Functions

The simplest input function in MATLAB is called **input**. The **input** function is used in an assignment statement. To call it, a string is passed that is the prompt that will appear on the screen, and whatever the user types will be stored in the variable named on the left of the assignment statement. For ease of reading the prompt, it is useful to put a colon and then a space after the prompt. For example,

```
>> rad = input('Enter the radius: ')
Enter the radius: 5
rad =
     5
```

Input Functions

If character or string input is desired, 's' must be added as a second argument to the **input** function:

```
>> letter = input('Enter a char: ','s')
Enter a char: g
letter =
g
```

If the user enters only spaces or tabs before hitting the Enter key, they are ignored and an ***empty string*** is stored in the variable:

```
>> mychar = input('Enter a character: ', 's')
Enter a character:
mychar =
     ''
```

Input Functions

However, if blank spaces are entered before other characters, they are included in the string. In the next example, the user hits the space bar four times before entering "go." The **length** function returns the number of characters in the string.

```
>> mystr = input('Enter a string: ', 's')
Enter a string:      go
mystr =
     go
>> length(mystr)
ans =
      6
```

Question

What would be the result if the user enters blank spaces after other characters? For example, the user here entered "xyz " (four blank spaces):

```
>> mychar = input('Enter chars: ', 's')
Enter chars: xyz
mychar =
xyz
```

Answer: The space characters would be stored in the string variable. It is difficult to see earlier, but is clear from the length of the string.

```
>> length(mychar)
 ans =
      7
```

The string can actually be seen in the Command Window by using the mouse to highlight the value of the variable; the xyz and four spaces will be highlighted.

Input

It is also possible for the user to type quotation marks around the string rather than including the second argument 's' in the call to the **input** function.

```
>> name = input('Enter your name: ')
Enter your name: 'Stormy'
name =
Stormy
```

or

```
>> name = input('Enter your name: ','s')
Enter your name: 'Stormy'
name =
'Stormy'
>> length(name)
ans =
     8
```

Input

```
>> num = input('Enter a number: ')
Enter a number: t
Error using input
Undefined function or variable 't'.

Enter a number: 3
num =
     3
```

MATLAB gave an ***error message*** and repeated the prompt. However, if *t* is the name of a variable, MATLAB will take its value as the input.

```
>> t = 11;
>> num = input('Enter a number: ')
Enter a number: t
num =
    11
```

Input

Separate **input** statements are necessary if more than one input is desired. For example,

```
>> x = input('Enter the x coordinate: ');
>> y = input('Enter the y coordinate: ');
```

Normally in a script the results from **input** statements are suppressed with a semicolon at the end of the assignment statements.

It is also possible to enter a vector. The user can enter any valid vector, using any valid syntax such as square brackets, the colon operator, or functions such as **linspace**.

```
>> v = input('Enter a vector: ')
Enter a vector: [3   8   22]
v =
     3     8    22
```

Output: disp and fprintf

Output statements display strings and/or the results of expressions, and can allow for ***formatting*** or customizing how they are displayed. The simplest output function in MATLAB is **disp**, which is used to display the result of an expression or a string without assigning any value to the default variable *ans*. However, **disp** does not allow formatting. For example,

```
>> disp('Hello')
Hello

>> disp(4 ^3)
     64
```

Output: disp and fprintf

Formatted output can be printed to the screen using the **fprintf** function. For example,

```
>> fprintf('The value is %d, for sure!\n',4 ^3)
The value is 64, for sure!
>>
```

To the **fprintf** function, first a string (called the ***format string***) is passed that contains any text to be printed, as well as formatting information for the expressions to be printed. In this case, the %d is an example of format information.

The %d is sometimes called a ***place holder*** because it specifies where the value of the expression that is after the string, is to be printed. The character in the place holder is called the ***conversion character***, and it specifies the type of value that is being printed. There are others, but what follows is a list of the simple place holders:

```
%d    integer (it stands for decimal integer)
%f    float (real number)
%c    character (one character)
%s    string of characters
```

Output: question

What do you think would happen if the newline character is omitted from the end of an **fprintf** statement?

Answer: Without it, the next prompt would end up on the same line as the output. It is still a prompt, and so an expression can be entered, but it looks messy as shown here.

```
>> fprintf('The value is %d, surely!', 4^3)
The value is 64, surely!>> 5 + 3
ans =
     8
```

Note that with the **disp** function, however, the prompt will always appear on the next line:

```
>> disp('Hi')
 Hi
 >>
```

Also, note that an ellipsis can be used after a string but not in the middle.

What would happen if you use the %d conversion character but you're trying to print a real number?

Answer: MATLAB will show the result using exponential notation

```
>> fprintf('%d\n',1234567.89)
1.234568e+006
```

Note that if you want exponential notation, this is not the correct way to get it; instead, there are conversion characters that can be used. Use the **help** browser to see this option, as well as many others!

Output: question

How can you get a blank line in the output?

Answer: Have two newline characters in a row.

```
>> fprintf('The value is %d,\n\nOK!\n',4^3)
The value is 64,

OK!
```

This also points out that the newline character can be anywhere in the string; when it is printed, the output moves down to the next line.

What do you think would happen if you tried to print 1234.5678 in a field width of 3 with 2 decimal places?

```
>> fprintf('%3.2f\n', 1234.5678)
```

Answer: It would print the entire 1234, but round the decimals to two places, that is,

1234.57

If the field width is not large enough to print the number, the field width will be increased. Basically, to cut the number off would give a misleading result, but rounding the decimal places does not change the number significantly.