



COMPUTER SKILLS

LESSON 6

Valeria Cardellini
cardellini@ing.uniroma2.it
A.Y. 2015/16

Objectives of this lesson

We'll discuss

- MATLAB scripts
- Input and output
- Scripts with input and output
- How to produce simple scripts

Algorithms

- Through MATLAB Command Window you can only perform simple calculation
 - We rather want to group statements together in a **computer program**
- Before writing the program we need to outline the **algorithm**: the sequence of steps needed to solve a problem
- Example: calculate the area of a circle
 - Get the input: the radius
 - Calculate the result: the area
 - Display the result: the area

Input and output

- Where does the **input** come from?
 1. **External file**
 2. **From the user**: we need **prompting**, i.e., telling the user what to enter
- One of the two is **the default input device**

- Where does the **output** go?
 1. **External file**
 2. **Screen**
- One of the two is **the default output device**

Programming languages

- Basic steps for programming
 - Analyze the **problem**
 - Write and refine the **algorithm** for its solution
 - We have already studied **flowcharts** to represent algorithms
 - **Code** the solution in a particular programming language
- Which programming languages?
 - **High-level languages**: have English-like commands and functions, e.g., MATLAB, Java, Python, C
 - **Low-level languages**, provide only little abstraction, e.g. Assembler
 - **Machine language**: the only language that the machine can execute directly without a previous transformation

Compiler or interpreter

- How to translate from high-level language to machine language? A special program (either a compiler or an interpreter) performs the task
- **Compiler**: translates the program written in high-level language (**source code**) to an executable program (**object code**) as a separate step
- **Interpreter**: goes through the code line by line, translating and executing each command as it goes
- MATLAB uses an interpreter
 - MATLAB code is interpreted rather than compiled

MATLAB scripts

- Scripts (rather than programs) in MATLAB terminology
 - Because of interpretation
- **Script**: sequence of MATLAB instructions that is store in a **M-file** (file with extension .m) and saved on the computer disk
 - Use the Editor to write the script
 - Check which is the current folder before saving the script (use **pwd** command in the Command Window)
 - pwd means “print working directory”
 - Save the script with the extension .m (it should be the default)
 - Use **type** command to display the script content in the Command Window (without .m)
 - Run or execute the script from the Command Window by entering the file name (without .m) at the prompt

The first script

- script1.m

```
radius = 5  
area = pi * (radius^2)
```

```
>> type script1 %show the contents of script1.m
```

```
radius = 5
```

```
area = pi * (radius^2)
```

```
>> script1 % execute the script
```

```
radius =
```

```
5
```

```
area =
```

```
78.5398
```


Documentation

- Document well your scripts so that people (and teachers!) can understand what the script does and how it accomplishes its task
- So put **comments** in the script!
 - Comments are ignored during the execution
- % symbol to put in the comment

Block comment

```
%This script calculates the area of a circle
```

```
% First the radius is assigned
```

```
radius = 5
```

```
% The area is calculated based on the radius
```

```
area = pi * (radius^2)
```

Input and output

- How to read the value of radius from an external source?
- How to print the output in a nice, informative way?
- Use **input/output (I/O) statements**

input function

- The simplest input function in MATLAB is **input**
- Use it in an assignment statement

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

```
Enter the radius: 5
```

```
rad =
```

```
5
```

- To input a character or string, use '**c**' or '**s**' as second argument to **input** function

```
>> mystr = input('Enter a string: ', 's')
```

- Empty string if the user enters only spaces or tabs
- Error message in case of unmatched data type, for example the user enters a character but the script was expecting a number

Output statements

- Output statements display strings and/or results of expressions and allows for **formatting** (i.e., customizing how they are displayed)
- The built-in function **disp** allows to display the result of an expression or a string, but does not allow for formatting

```
>> disp('Hello world!')
```

```
Hello
```

- The built-in function **fprintf** allows to print to the screen formatted output

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

```
The value is 64, for sure!
```

```
>>
```

fprintf function

Input arguments of the **fprint** function:

- **Format string**: any text to be printed as well as formatting information for the expressions to be printed
- **Place holder (%) and conversion character**
 - **%d** for integer values
 - **%f** for float values
 - **%c** for single character
 - **%s** for string
- Newline character '**\n**'
- A **field width** can also be included in the place holder to specify how many characters are to be used

```
>> fprintf('Greek pi value is %.2f (%f)\n', pi, pi)
Greek pi value is 3.14 (3.141593)
>>
```

Printing vectors

- For printing vectors and matrices, **disp** can be easier to use than **fprintf**
- **fprintf** for a vector
 - If a conversion character and the newline character are in the format string, the vector will print in a column
 - Without the newline, it would print in a row

```
>> v = 1:3:12
```

```
>> fprintf('%d ', v);
```

```
1 4 7 10>>
```

```
>> fprintf('%d\n', v)
```

```
1
```

```
4
```

```
7
```

```
10
```

```
>> disp(v)
```

```
1
```

```
4
```

```
7
```

```
10
```

Printing vectors and matrices

- **fprintf** for a matrix
 - Specifying only one conversion character and the newline character, the elements of the matrix will be printed in one column
 - Specifying the number of elements in each row, the matrix will be printed column by column

```
>> v1 = 1:2:6;
```

```
>> v2 = [4,7,9];
```

```
A = [v1; v2];
```

```
>> fprintf('%d ', A)
```

```
1 4 3 7 5 9>>
```

```
>> fprintf('%d %d %d\n', A)
```

```
1 4 3
```

```
7 5 9
```

```
>> fprintf('%d\n', A)
```

```
1
```

```
4
```

```
3
```

```
7
```

```
5
```

```
9
```

Extract the elements of a vector with odd index

