Unit 5: Subprograms

uc3m | Universidad Carlos III de Madrid

DEI Interactive Systems Group

Objectives

- Understand the need of organizing the code in scripts and functions
- Learn to write functions and scripts in MATLAB
- Understand how parameters are passed to a function
- Know the different types of variables

Programme modules

- Programs may be too long and complex to write as a single unit.
- A program can be logically divided in smaller subprograms or modules
- Advantages:
  - Divide and Rule
  - Easier to maintain and to debug
  - Easier to reuse

Modular Programming in MATLAB

- There are two kinds of M-files (*.m):
  - Scripts:
    - They operate on data in the main workspace.
    - Do not accept input arguments or return output arguments.
  - Functions:
    - Internal variables are local to the function
    - They can accept input arguments and return output arguments.

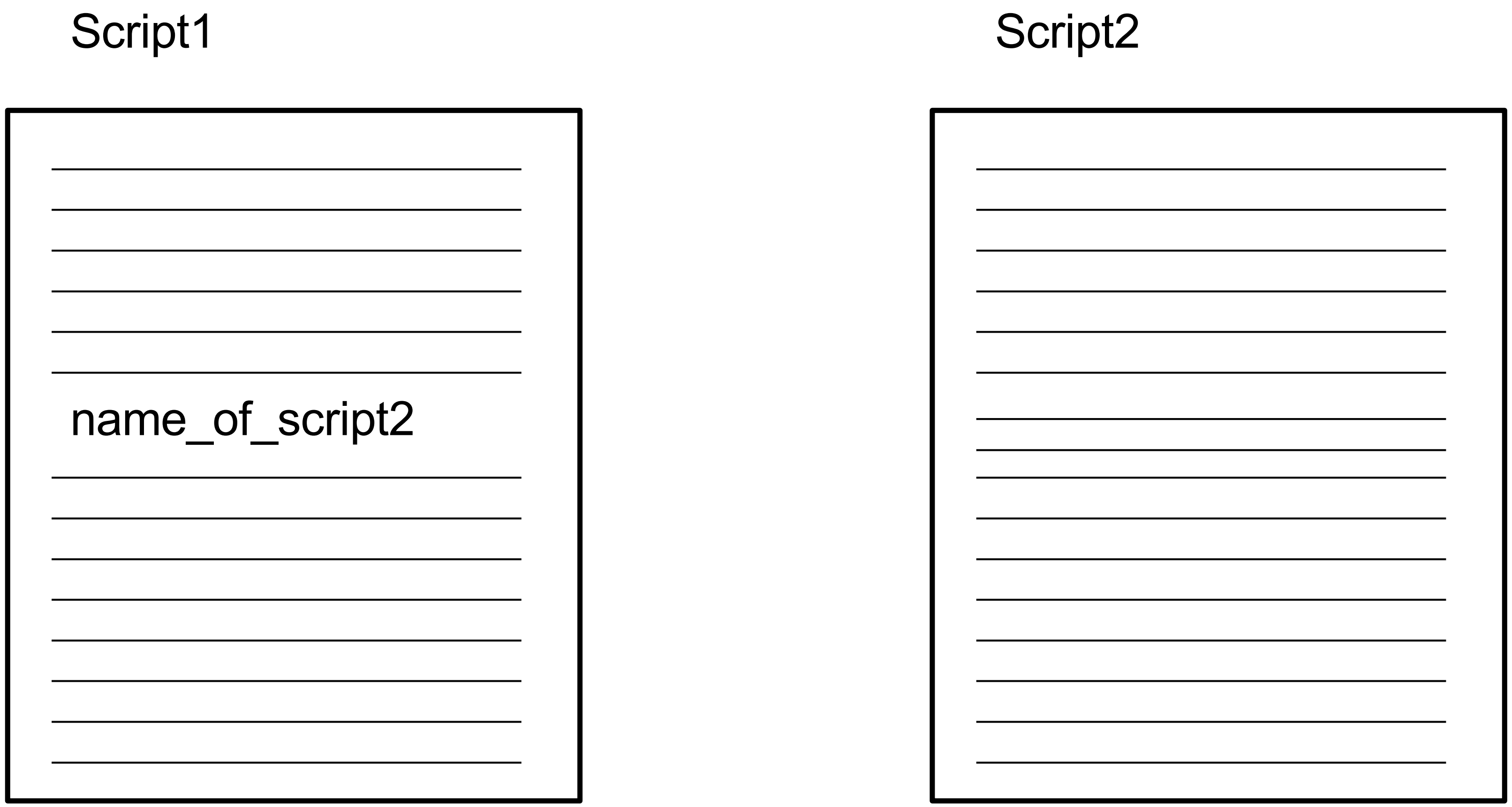
Scripts or m-files

- Contain sequences of MATLAB commands
- Whenever a command produces an output the result is visualized in the Command Window
- They can be executed from...

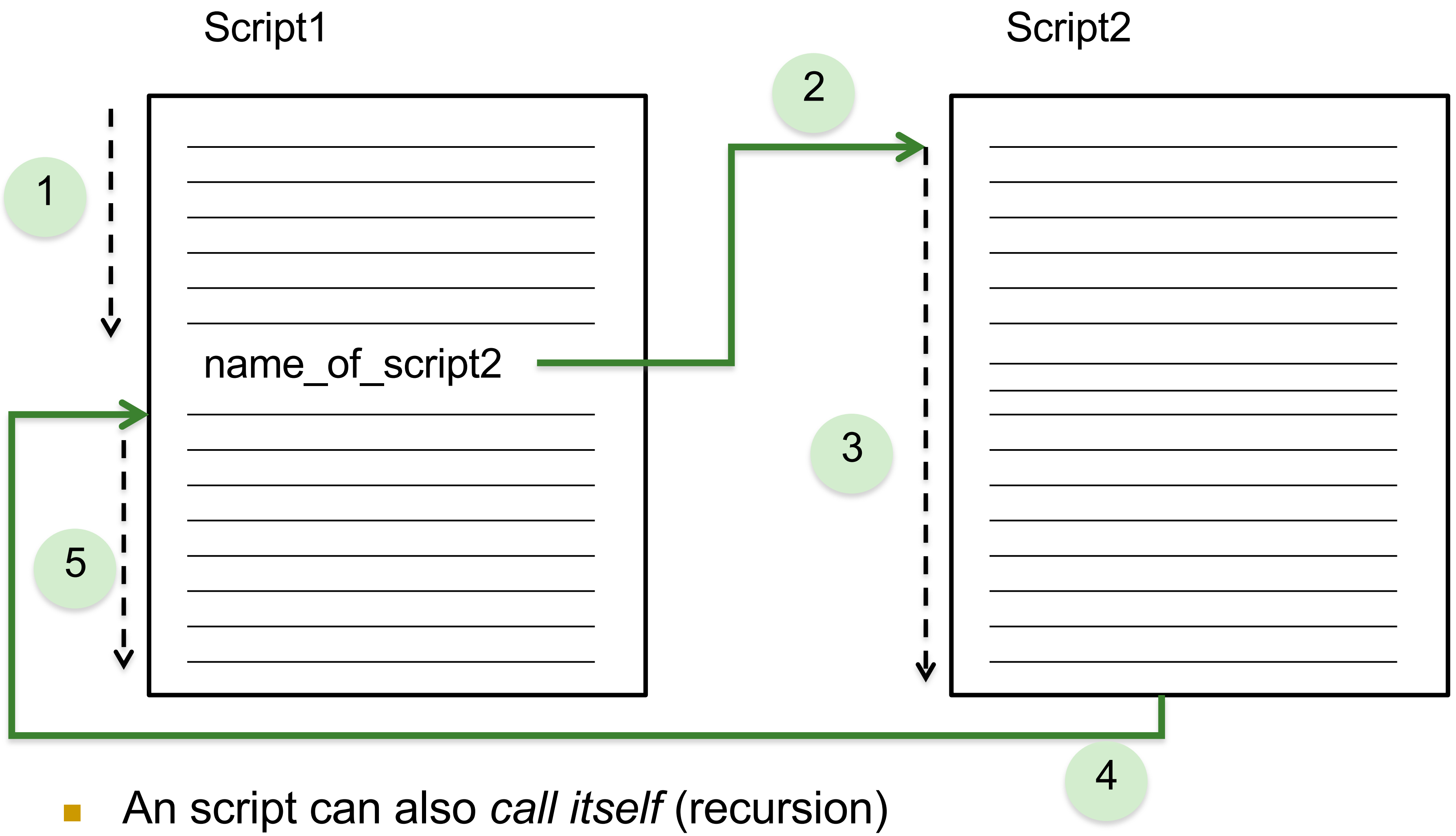
Scripts or m-files

- Contain sequences of MATLAB commands
- They can be executed from...
  - Command window: typing the name of the script
  - MATLAB editor: using the ‘run’ icon
  - Other script: including the name of the script in the sequence of commands (*calls*)

Scripts or m-files: calls



Scripts or m-files: calls



- An script can also *call itself* (recursion)

Scripts or m-files: variables

- Scripts can operate on existing variables of the workspace, or they can create new variables on which to operate.
- When a script is executed from the command window or called from another script the variables created belong to the MATLAB workspace

Scripts or m-files. Example of CALL

```
% Script example 1
driver = ‘f’;
years = 27;
scriptExample2;
fprintf(‘the dto is %d’, dto);
```

```
% Script example 2
if (driver == ‘f’) &(years < 30)
        dto = 10;
else
        dto = 5;
end;
```

MATLAB WORKSPACE

Scripts or m-files. Example of CALL

```
% Script example 1
driver = 'f';
years = 27;
scriptExample2;
fprintf('the dto is %d', dto);
```

```
% Script example 2
if (driver == 'f') &(years < 30)
        dto = 10;
else
        dto = 5;
end;
```

MATLAB WORKSPACE

Variable name →	**driver**	**years**
Variable value →	f	27

Scripts or m-files. Example of CALL

```
% Script example 1
driver = 'f';
years = 27;
scriptExample2;
fprintf('the dto is %d', dto);
```

```
% Script example 2
if (driver == 'f') &(years < 30)
        dto = 10;
else
        dto = 5;
end;
```

MATLAB WORKSPACE

Variable name	**driver**	**years**
Variable value	f	27

Scripts or m-files. Example of CALL

```
% Script example 1
driver = 'f';
years = 27;
scriptExample2;
fprintf('the dto is %d', dto);
```

```
% Script example 2
if (driver == 'f') &(years < 30)
        dto = 10;
else
        dto = 5;
end;
```

MATLAB WORKSPACE

Variable name →	**driver**	**years**	**dto**
Variable value →	f	27	10

Scripts or m-files. Example of CALL

```
% Script example 1
driver = ‘f’;
years = 27;
scriptExample2;
fprintf(‘the dto is %d’, dto);
```

```
% Script example 2
if (driver == ‘f’) &(years < 30)
        dto = 10;
else
        dto = 5;
end;
```

MATLAB WORKSPACE

Variable name →	**driver**	**years**	**dto**
Variable value →	f	27	10

Scripts or m-files. Example of CALL

```
% Script example 1
driver = 'f';
years = 27;
scriptExample2;
fprintf('the dto is %d', dto);
```

```
% Script example 2
if (driver == 'f') &(years < 30)
        dto = 10;
else
        dto = 5;
end;
```

MATLAB WORKSPACE

Variable name →	**driver**	**years**	**dto**
Variable value →	f	27	10

Scripts or m-files: variables

- Useful commands:
  - `echo on/off`: when activated (`echo on`) prints the commands in the script as they are executed.
    - This can be very useful when debugging (finding errors) in our programs
  - `help` *`scriptname`*: shows the first two lines of comments of the script. Useful for documenting.
  - `clear`: cleans the workspace, removing all the existing variables.
    - It is a good practice to put the word clear at the beginning of a script to make sure variables from previous executions of other programs do not interfere.

Functions

- MATLAB allows users to create their own functions, which will work in a similar way to the MATLAB functions *rem*, *floor*, *sqrt*..
- User functions are defined in m-files in a similar way to the scripts but following a specific syntax
- User functions can be called from the command window, from scripts or from other functions
    - Each call must supply values for the input arguments of the function and retrieve the values of the output arguments
- Every function has its <u>own *function workspace*</u>

Functions. Example

The code is stored in a file named *obtainSalary.m*

```
function [salary]= obtainSalary(wage,hoursWorked)
% Function to compute the salary of a worker
% Extra hours are paid a 50% more
% wage = wage of the worker in euros
% hoursWorked = hours worker per week
% extra = extra salary of the worker per week
% salary = salary of the worker per week

base = wage * hoursWorked;
if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
else
      extra = 0;
end
salary = base + extra;
end
```

Function definition

```
function [salary]= obtainSalary(wage,hoursWorked)
% Function to compute the salary of a worker
% Extra hours are paid a 50% more
% wage = wage of the worker in euros
% hoursWorked = hours worker per week
% extra = extra salary of the worker per week
% salary = salary of the worker per week

base = wage * hoursWorked;
if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
else
      extra = 0;
end
salary = base + extra;
end
```

Function definition

H1 line

Help text

Body of the function

Function definition

```
function [salary]= obtainSalary(wage,hoursWorked)
% Function to compute the salary of a worker
% Extra hours are paid a 50% more
% wage = wage of the worker in euros
% hoursWorked = hours worker per week
% extra = extra salary of the worker per week
% salary = salary of the worker per week

base = wage * hoursWorked;
if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
else
      extra = 0;
end
salary = base + extra;
end
```

Function definition

H1 line

Help text

Body of the function

Function definition

- Function definition
  - First line of the code
  - Indicates that the file contains a function
  - Defines the function name
  - Defines the number and order of the input and output parameters

```
function [output arguments]= function_name (input paramaters)
```

Function keyword. Always in lowercase

List of output arguments separated by comas. **A function may have from 0, 1 or more than 1 output parameters**

Name of the function. Can be composed of characters, digits and the underscore symbol. **Use the same function_name as the m file that contains the code**

List of input parameters separated by commas. **A function may have from 0, 1 or more than 1 input parameters**

Function definition

```
function [salary]= obtainSalary(wage,hoursWorked)
% Function to compute the salary of a worker
% Extra hours are paid a 50% more
% wage = wage of the worker in euros
% hoursWorked = hours worker per week
% extra = extra salary of the worker per week
% salary = salary of the worker per week

base = wage * hoursWorked;
if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
else
      extra = 0;
end
salary = base + extra;
end
```

Function definition

H1 line

Help text

Body of the function

Function definition

- The H1 line and help lines are comments which makes it easy to document your functions
- H1 line:
  - First comment line of the function
  - Normally contains the name of the function and a brief description
  - When a user types `lookfor` *`word`* in the command window MATLAB retrieves all the H1 lines which contains that word
- Help lines:
  - Comment lines between the H1 line and the first line of code
  - The command `help` *`function_name`* retrieves the help lines of that function

Function definition

```
function [salary]= obtainSalary(wage,hoursWorked)
% Function to compute the salary of a worker
% Extra hours are paid a 50% more
% wage = wage of the worker in euros
% hoursWorked = hours worker per week
% extra = extra salary of the worker per week
% salary = salary of the worker per week

base = wage * hoursWorked;
if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
else
      extra = 0;
end
salary = base + extra;
end
```

Function definition

H1 line

Help text

Body of the functior

The word *end* marks the end of the body of the function.
Do not put a ; at the end of this line or you will get an error

Functions. Example of call

exercise1.m

```
   yourWage = input('Introduce your wage');
   yourHours  = input('Introduce the hours worked');
   yourSalary = obtainSalary(yourWage, yourHours);
   fprintf('Your salary is %d', yourSalary);
```

obtainSalary.m

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end
   salary = base + extra;
   end
```

Functions. Example of call

```
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
```

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

The values of the variables *yourWage* and *yourHourse* are copied to the function's variables (arguments) *wage* and *hoursWorked*

Functions. Example of call

```
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
```

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

When Matlab finishes executing the function the value of the variable *salary* is copied to the variable *yourSalary.*

(Matlab does this because the name of variable salary appears in the list of output variables in the function definition line)

Functions. Example with 2 output values

```
function [salary, extra]= obtainSalary(wage,hoursWorked)
% Function to compute the salary of a worker
% Extra hours are paid a 50% more
% wage = wage of the worker in euros
% hoursWorked = hours worker per week
% extra = extra salary of the worker per week
% salary = salary of the worker per week

base = wage * hoursWorked;
if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
else
      extra = 0;
end;
salary = base + extra;
end
```

Functions. Example with 2 output values. Call

```
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
[yourSalary, yourExtra] = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
fprintf('Your extra  is %d', yourExtra);
```

Functions. Example with 2 output values. Call

```
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
[yourSalary, yourExtra] = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
fprintf('Your extra  is %d', yourExtra);
```

```
function [salary, extra]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Passing parameters to a function

- There are two different ways in which a **programming language** may implement the calls of functions:
  - **Pass by values**: The values of the variables of the calling code are copied into the variables specified as parameters of the functions.
    - **Changes in these variables do not modify the values of the variables of the calling code.**
  - **Pass by reference**: Changes in the variables of the function modify the values of the variables passed as parameters in the calling code.

In MATLAB calls to all functions use **pass by values**

Local Variables

- Local variables:

Local Variables

- Local variables:
  - **Every function has its own *function workspace*** separated from the workspace used by the command window and the scripts, and the workspaces of the rest of the functions.

Local Variables

- Local variables:
  - **Every function has its own *function workspace*** separated from the workspace used by the command window and the scripts, and the workspaces of the rest of the functions.
  - **Variables defined in** the body of a **function** (including input and output arguments) **are only recognized inside the function** scope.

Local Variables

- Local variables:
  - **Every function has its own *function workspace*** separated from the workspace used by the command window and the scripts, and the workspaces of the rest of the functions.
  - **Variables defined in** the body of a **function** (including input and output arguments) **are only recognized inside the function** scope.
  - Once the execution of the function finished **its workspace is eliminated and the current value of the variables is lost**. If the function is called again a totally new workspace will be created

Local Variables

```
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
```

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

salary, wage, *hoursworked*, *base* and *extra* are **LOCAL VARIABLES**. They belong to the **function workspace**

yourwage, *yourHours*, and *yourSalary* are **NOT LOCAL**. They belong to the **Matlab workspace**

Local Variables

STEP 1. The user introduces his/her wage and hours

```
yourWage = input('Introduce your wage');            ← 500
yourHours  = input('Introduce the hours worked');   ← 20
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
```

MATLAB WORKSPACE

yourWage	yourHours
500	20

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Local Variables

STEP 2. The function obtain salary is called

```
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
```

MATLAB WORKSPACE

yourWage	yourHours
500	20

FUNCTION WORKSPACE

wage	hoursWorked
500	20

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Local Variables

STEP 3. The function obtain salary is executed

```
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
```

MATLAB WORKSPACE

yourWage	yourHours
500	20

FUNCTION WORKSPACE

wage	hoursWorked	base	salary	extra
500	20	1000	1000	0

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Local Variables

STEP 4. The execution of the function finished and Matlab continue on and prints the salary on screen

```
yourWage = input(‘Introduce your wage’);
yourHours  = input(‘Introduce the hours worked’);
yourSalary = obtainSalary(yourWage, yourHours);
fprintf(‘Your salary is %d’, yourSalary);
```

MATLAB WORKSPACE

yourWage	yourHours	yourSalary
500	20	1000

FUNCTION WORKSPACE

DELETED

wage			salary	extra
500	20	1000	1000	0

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Local Variables

```
yourWage = input('Introduce your wage');          500
yourHours  = input('Introduce the hours worked'); 20
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
fprintf('Value of salary %d', salary);
```

If we add this last line… what MATLAB will print on screen?

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Local Variables

```
yourWage = input('Introduce your wage');          ← 500
yourHours  = input('Introduce the hours worked'); ← 20
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
fprintf('Value of salary %d', salary);
```

```
function [salary]= obtainSalary(wage,hoursWorked)
        unction to compute the salary of a worker
        xtra hours are paid a 50% more
        age = wage of the worker in euros
        oursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Matlab will give an **error** as the variable 'salary' does not exists in its workspace

MATLAB WORKSPACE

yourWage	yourHours	yourSalary
500	20	1000

Local Variables

```
salary = 0;
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
fprintf('Value of salary %d', salary);
```

And if we add the first and last line… what MATLAB will print on screen?

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
   if (hoursWorked > 40)
      extra = (hoursWorked-40) * wage /2;
   else
      extra = 0;
   end;
   salary = base + extra;
   end
```

Local Variables

```
salary = 0;
yourWage = input('Introduce your wage');
yourHours  = input('Introduce the hours worked');
yourSalary = obtainSalary(yourWage, yourHours);
fprintf('Your salary is %d', yourSalary);
fprintf('Value of salary %d', salary);
```

500

20

The *salary* variable of the script is not the same as the *salary* variable defined in the function body

MATLAB WORKSPACE

salary	yourWage	yourHours	yourSalary
0	500	20	1000

```
function [salary]= obtainSalary(wage,hoursWorked)
 % Function to compute the salary of a worker
 % Extra hours are paid a 50% more
 % wage = wage of the worker in euros
 % hoursWorked = hours worker per week
 % extra = extra salary of the worker per week
 % salary = salary of the worker per week

   base = wage * hoursWorked;
     (hoursWorked > 40)
     extra = (hoursWorked-40) * wage /2;
    e
     extra = 0;
   end;
   alary = base + extra;
   nd
```

Matlab will print:
Value of salary 0

drid

Exercise

- Write a function called ‘hypothenuse’ that receives as parameters the lengths of two sides of a triangle and returns the value of the hypotenuse.
  - To compute the square root you can use the function *sqrt*

Exercise

- Write a function called ‘hypothenuse’ that receives as parameters the lengths of two sides of a triangle and returns the value of the hypotenuse.

```
function [hyp] = hypothenuse(sideA, sideB)
 % function hypothenuse
 % Given the two sides of a triangle computes its
  % hypothenuse
 hyp = sqrt(sideA^2+sideB^2);
end
```

Exercise

- Write a program that asks the user to introduce the length of two sides of the triangle, calls the hypothenuse function and prints the result on screen

Exercise

- Write a program that asks the user to introduce the length of two sides of the triangle, calls the hypothenuse function and prints the result on screen

```
varSideA = input ('Introduce the lenght of one side');
varSideB = input ('Introduce the lenght of the other side');
varHypo = hypothenuse (varSideA, varSideB);
fprintf('\n The hypthenuse is %d', varHypo);
```

Exercise

- Write a function ‘obtainSeconds’ that receives as parameters three numbers representing hours, minutes and seconds and returns the total number of seconds.

Exercise

- Write a function ‘obtainSeconds’ that receives as parameters three numbers representing hours, minutes and seconds and returns the total number of seconds.

```
function [totalseconds] = obtainSeconds(hours, minutes, seconds)
    % function obtainSeconds
    % Receives a number of hours, minutes and seconds and return
    % the total number of seconds
    totalseconds = hours*3600+minutes*60+seconds;

end
```

Exercise

- Write a function ‘obtainTime’ that receives a number representing a total of seconds and returns the corresponding hours, minutes and seconds.
- Test the function writing a program that asks the user to introduce a number of seconds and prints on screen the corresponding hours, minutes and seconds.

Exercise

PROGRAM

```
vtotalSeconds = input('Introduce seconds);
[varHours, varMinutes, varSeconds] = obtainTime(vtotalSeconds);
fprintf('Hours: %d \t Minutes: %d \t Seconds %d', varHours,
   varMinutes, varSeconds);
```

Function

```
function [hours, minutes, seconds] = obtainTime(totalSeconds)

 hours = floor(totalSeconds / 3600);
 restSeconds = rem(totalSeconds,3600);
 minutes = floor(restSeconds/ 60);
 seconds = rem(restSeconds, 60);

end
```

Exercise

- Write a function ‘perfect’ that receives a number and returns 1 (true) if the number is perfect and 0 if it is not. A number is perfect when the sum of its factors (excluding the number) is equal to its value.
  - Examples:
    - 6 is perfect as its factors are 1, 2 and 3 and 1+2+3 = 6
    - 28 is perfect as 1+2+4+7+14 = 28
  - Write a program that makes use of the function for printing on screen all the perfect numbers between 1 and 1000.

Exercise (function)

```
function [ rdo ] = perfect( varNumber )
    sumFactors = 0;
    maxFactor = floor(varNumber/2);
    for i=1:maxFactor
        if rem(varNumber,i) == 0
            sumFactors = sumFactors + i;
        end;
    end;
    if (varNumber == sumFactors)
        rdo = 1;
    else
        rdo = 0;
    end;
end
```

Exercise (Program)

```
for i=1:1000
        if perfect(i) == 1
                fprintf('\n%d',i);
        end;
end;
```

Exercise (Program)

This also works. Remember that perfect(i) is going to return 1 (true) or 0 (false)

```
for i=1:1000
        if perfect(i)
                fprintf('\n%d',i);
        end;
end;
```

Remember

- Write each function in a separate file. Only one function per file
- The function and the file should have the same name
- Do not put a ; after the keyword *end* at the end of the function
- The variables in the function are local. You can’t access them from the other functions or script