MATLAB for Psycologists

CdL Scienze e Tecniche Psicologiche
a.a. 2018-2019

ANDREA FROSINI
e-mail: andrea.frosini@unifi.it

Testo di riferimento:
M. Borgo, A. Soranzo, M. Grassi
“MATLAB for Psycologists”
Springer

errata corrige, scripts and listings from the book:
https://dpg.unipd.it/en/mlp/matlab-book

Chapter 1– Basic Operations

Step zero: get used to the environment, create a directory where save and open your file(s).

Basic arithmetical operators:

To type after prompt >> followed by Enter	MATLAB answer	meaning
35+12	ans = 47	sum
35*12	ans = 420	multiplication
2/45	ans = 0.0444	division
2-1	ans = 1	subtraction
2^3	ans = 8	exponentiation
12/0	ans = Inf	infinity
0/0	ans = NaN	Not a Number
11+	??? 11+	expression error

Chapter 1– Variables

A variable can be regarded as a labeled box having a prescribed dimension which contains a *certain type of data* (automatically created and dynamically updated according to the context)

Create, update and recall a variable:

>>pippo=9 Ans= 9	>> pippo Ans= 9
>>pippo='ciao mamma' Ans= ciao mamma	>>pippo Ans= ciao mamma

Task: enjoy updating variables and make arithmetical operations on them. Keep track of their changes in the Workspace.

N.B. Using semicolon ; at the end of a command prevents the command to be echoed on the screen

Chapter 1– Vectors and Matrices

Let us create vectors and matrices and recall them:

```
>>a=[1,2,3,4];
>>b=[1;2;3];
>>c=[1,2,3,4;5,6,7,8;9,10,11,12];
```

Vectors' dimension changes!!!

a (row) vector	b (column) vector	c 3x4 matrix	Assign value	ATTENTION!
1 2 3 4	1 2 3	1 2 3 4 5 6 7 8 9 10 11 12		>>d=5 d=5
>>a(2) Ans= 2	>>b(0,1) error	>>c(3,2) Ans= 10	>>a(1,2)=6 Ans= **a**= 1 6 3 4	>>d(2)=6 d= 5 6

N.B. the followings are useful functions

>>size(c) Ans= 3 4	**>>length(a) Ans= 4**	**>>length(b) Ans= 1**

Chapter 1– Vectors and Matrices (nice tricks)

Address more than one element at a time:

>>c([1,3];4)	>>c(2;1:3)	>>c(:;1)	>>c(:,[2,4]) = []
1 2 3 **4** 5 6 7 8 9 10 11 **12**	1 2 3 4 **5** **6** **7** 8 9 10 11 12	**1** 2 3 4 **5** 6 7 8 **9** 10 11 12	1 **2** 3 **4** 5 **6** 7 **8** 9 **10** 11 **12**
Ans= 4 12	Ans= 5 6 7	Ans = 1 5 9	Ans = **c**= 1 3 5 7 9 11

Useful functions and operations: let d=[2,4;5,7;9,11;1,0], e=[7,8,9,0]

>>size(c) Ans= 3 4	>>length(c) Ans= 4	>>length(a) Ans= 4	
>>2*a Ans= 2,4,6,8	>>a+e Ans= 8 10 12 4	>>c*d Ans=43 51 111 139 179 227	>>d' % transposition Ans = 2 5 9 1 4 7 11 0

Exercises 1,2 and 3 are suggested

Chapter 2- Data Handling

MATLAB stores logical values and strings in addition to numbers into variables, with simple or structured data types.

Handling Logical Variables:

A logical variable stores the two logical data

FALSE represented by 0;

TRUE represented by any nonzero (usually 1) number.

The function *logical(x)* converts the elements of the vector *x* into logical values

The *relational operators* that can be used in MATLAB are:

< (less) ,<= (less or equal), > (greater),>= (greater or equal),== (equal),~= (not equal)

The *logical operators* that can be used in MATLAB are:

& (AND) , | (OR),~ (NOT)

Chapter 2- Data Handling

Examples of the use of logical variables and operators **(TO BE UNDERSTOOD):**

let a=[0,1,2,3,4], b=[3,2,0,1,7]

>>5>3 Ans= 1	>>logical(a) Ans= 0 1 1 1 1	>>a>b Ans= 0 0 1 1 0	>>c= a==3 c= 0 0 0 1 0	>>x=3; 0<x<2 Ans= 1	>>(x>0)&(x<3) Ans= 0
>>c=(b>=3)|(b<1) c= 1 0 1 0 1 >>d=b(c) d= 3 0 7		>>e= a((b>=3)|(b<1)) e = 0 2 4		>>any(a) Ans = 1 >>f=[0,0,0]; any(f) Ans = 0	>>all(a) Ans= 0 >>f=[1,2,3];all(f) Ans=1
>>exist('a') Ans = 1 >>exist('z') Ans=0		>>isempty(a) Ans = 0 >>f=[];isempty(f) Ans = 1		Etc…	Etc…

N.B. the example in cell (1,5) needs a further comment: MATLAB resolves the command 0<x<2 as follows

1) it computes 0<x that is true so it gives 1 as result;

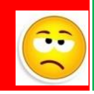
2) it computes 1<2 that is true, giving 1 as Ans.

Chapter 2- Data Handling

Handling Strings:

A string is a sequence of characters and are treated by MATLAB as vectors

If we need a sequence of strings, then we have to use the function *char* that creates a matrix of strings, each in a different row

>>a='Mario' a= Mario >>a(2) Ans= a	>>b='Luigi' >>c=[a,' ',b] c= Mario Luigi	>>c=a; c(2)=b Error 	>>c=char(a,b) c= Mario Luigi 	>>lower(a) Ans = mario >>upper(a) Ans= MARIO
>>strcmp(a,b) Ans= 0 >>strcmp(a,c(1)) Ans=1	>>strrep(a,'a','b') Ans=Mbrio % replace the occurrences of 'a' with 'b' in a		>>findstr(b,'i') Ans= 3 5 % return the indexes of each occurrence of 'i' in b	

Chapter 2– Data Handling

Handling (formatted) Strings:

Data values or variables can be inserted into a string:

let a=‘Mario’, b=‘Luigi’,eta=[21,22]

>>sprintf(‘Il nome del mio amico e’’ %s ed ha %d anni’,a,eta(1)) Ans= Il nome del mio amico e’ Mario ed ha 21 anni >>sprintf(‘Il nome del mio amico e’’ %s ed ha %d anni’,b,eta(2)) Ans= Il nome del mio amico e’ Luigi ed ha 22 anni	Special characters spec.: %c – single char %d – integer number %s – string of chars %f – decimal number \n – newline \t – horizontal tab

INPUT		
>>input (‘How old are you? ’) How old are you? 35 Ans = 35	>>input(‘How old are you?’,’s’) How old are you? Thirty five Ans = Thirty five % ‘s’ is for string inputs	>>a= input(‘Name a friend ’,’s’) Name a friend Luca a = Luca

Chapter 2- Data Handling

Handling NaN:

NaN means *Not a Number* and is used for missing data.

Doing mathematical operations involving NaN return NaN.

>>pippo=[12, NaN, 5, NaN, 0, 3] Pippo = 12 NaN 5 NaN 0 3 >>isnan(pippo) Ans = 0 1 0 1 0 0 % isnan(pippo) return an array with 1 in NaN positions of pippo, 0 otherwise	>>mean(pippo) Ans = NaN >>mean(pippo(~isnan(pippo)) Ans= 5

Chapter 2- Data Handling

Handling Structures:

Structures are *structured data types* that can be regarded as vectors of different *primitive* (i.e., numbers, boolean and strings) data types.

Each element is called *field.* As usual, examples will clarify the use; let us assume we want to store the partecipants of an experiment:

```
>>subject.name='Mario'
>>subject.surname='Rossi'
>>subject.age=24
>>subject.testanswers=[2,1,4,1,2]
>>subject.testcorrections=logical([1,0,0,1,1])
>>subject
Subject=
    name: 'Mario'
    surname: 'Rossi'
    age : 24
    testanswers : [2,1,4,1,2]
    testcorrections :  1 0 0 1 1
```

```
>>subject(2).name='Luigi'
>>subject.age=20
>>subject.testanswers=[1,1,2,2,2]
>>subject(2)
Ans =
    name: 'Luigi'
    surname: []
    age : 20
    testanswers : [1,1,2,2,2]
    testcorrections :
>>rmfield(subject,'testanswers')
Ans = name
       surname
      testanswers
      testcorrections
```

Chapter 2- Data Handling

Handling Cells (skipped).

Import/Export:

In MATLAB it is extremely useful to save and load variables to and from files for further working sessions, since the program deletes them as soon as you quit.

>>clear all % clear all the variables in the workspace >>a='Mario' >>b='Luigi' >>c=[12 3 5 NaN] >>save pippo % creates in the working directory the file pippo.mat storing the current values of the three variables a,b and c	>>a='Luca' >>load pippo >>a a = Mario >>uiimport % import the data from a selected file	>>a='Luca' >>b='Camilla' >>save pippo a b % update in pippo.mat only the variables a and b

Exercises 1and 2 are suggested

Check carefully the code for the mean at pg.45

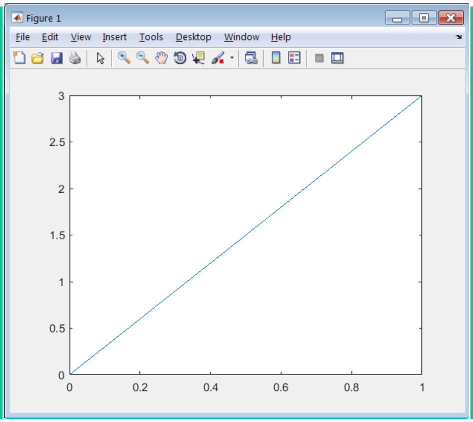
Chapter 3- Plotting Data

MATLAB plots data in many different ways:

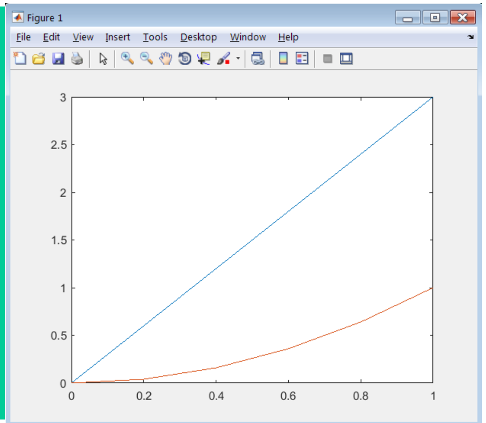
the simplest (and the only one considered here) is the *plot* command that inputs two sequences (x1,x2,…,xn) and (y1,y2,…,yn) of numbers and draws the polyline connecting (x1,y1),(x2,y2),…,(xn,yn).

Useful hints: x=[0:0.2:1] lists all the values from 0 to 1 with 0.2 step, i.e., x=[0,0.2,0.4,0.6,0.8,1], y=3*x creates a vector y=[0,0.6,1.2,1.8,2.4,3]

```
>>x=[0:0.2:1]
x= 0  0.2  0.4  0.6  0.8  1
>>y=3*x
y=0  0.6  1.2  1.8  2.4  3
>>plot(x,y)
```




```
>>plot(x,y,x,x.*x)
% .* is the element by element product in vectors
```



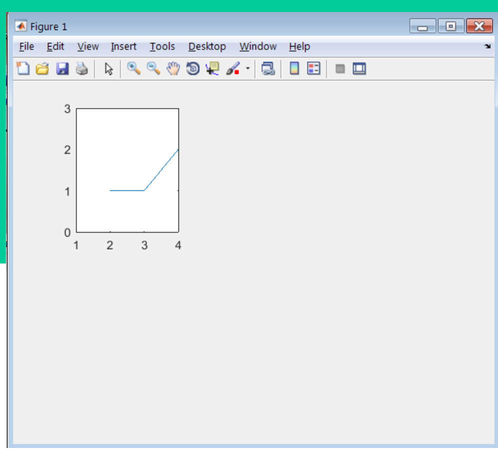
There are several plot command options to set the appearance of the figure: line aspect, line color, axes width, legenda, title, labels etc… They will be treated if needed.

Chapter 3– Plotting Data

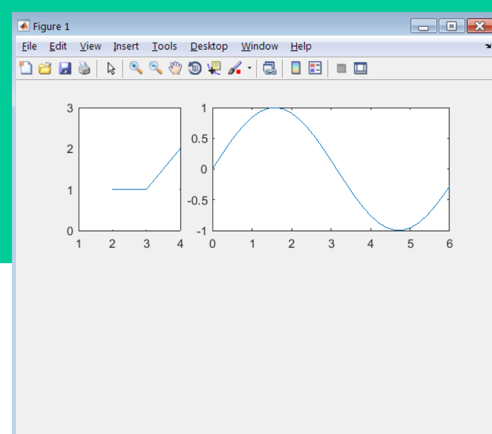
To display multiple graphics in one figure one can use the *subplot* sommand.

The figure area is considered as a matrix and each draw is placed where desired, accordingly. The syntax of the command is >>subplot(Nrows,Ncolumns,Position), where Nrows and Ncolumns are the rows and columns of the matrix division of the area, and Position is the area where the plot has to be placed. Areas are numbered from top to bottom and from left to right.

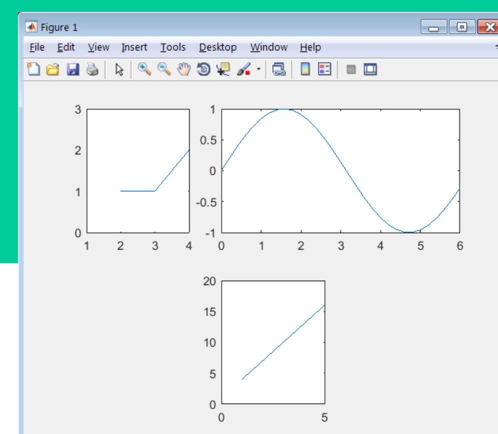
```
>>subplot(2,3,1)
>>plot([2,3,4],[1,1,2])
>>axes([1,4,0,3])
```




```
>>subplot(2,3,[2,3])
>>x=[0:0.2,6]
>>plot(x,sin(x))
```




```
>>subplot(2,3,5)
>>x=[1:0.2:5]
>>plot(x,3.*x+1)
```

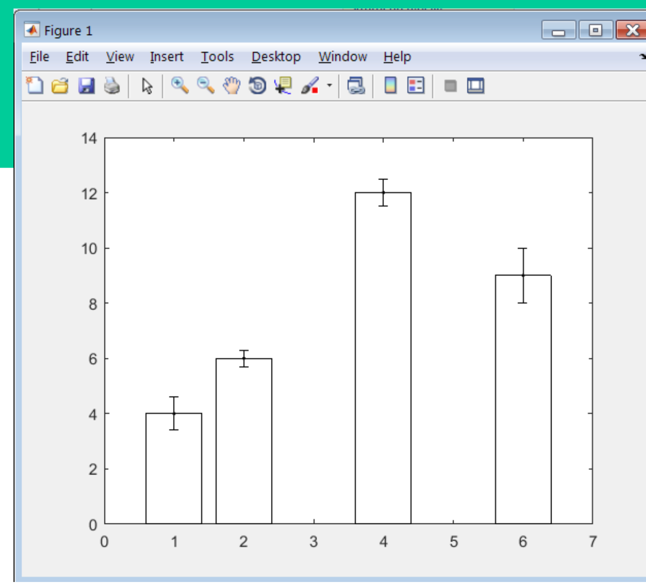


Chapter 3- Plotting Data

Common ways to display data: let rand(r,c) creates a *r x c* matrix with random data in [0,1] interval

```
>>bar(rand(5,3),'stacked')
>>colormap(winter)
>> title('candies in five markets')
>> set(gca,'XTickLabel',{'jan','feb','mar'})
>> set(gca,'YTick',[1,2])
```


```
>>x=[1,2,4,6]; values=[4,6,12,9]
>>dev=[0.6,0.3,0.5,1]
>> bar(x,values,'w'); hold on;
>> errorbar(x,values,dev,'.k')
```



Chapter 3- Plotting Data

Useful hint:

if one needs a matrix (vector) with integer random values in the interval [1,n], use the command

`>>fix(mod(rand(r,c).*10*n,n))`

where *fix* returns the integer part of a number, and *mod(x,y)* returns the remainder after division of *x* by *y*

After studying this chapter be also AWARE of:

-different kinds of graph representations

-how to change graph properties using *set* command

-3D data representations

-use of *hold on* command (do not allow a plot to replace the previous)

-use of *print* command

Exercises at will

Chapter 3– Plotting Data

Exercise:

store in a struct variable *test* the results of a five Lickert levels – six items test obtained from 12 subjects together with their name, surname and gender (use the random generator to obtain the results).

Then show the following three graphs at the same time:

– for each item, the number of each result by a bar in a six bar bargraph;

– the polyline of the total points obtained by the 12 subjects;

– the mean and the sd of the obtained results using the errorbar in a six bar bargraph .

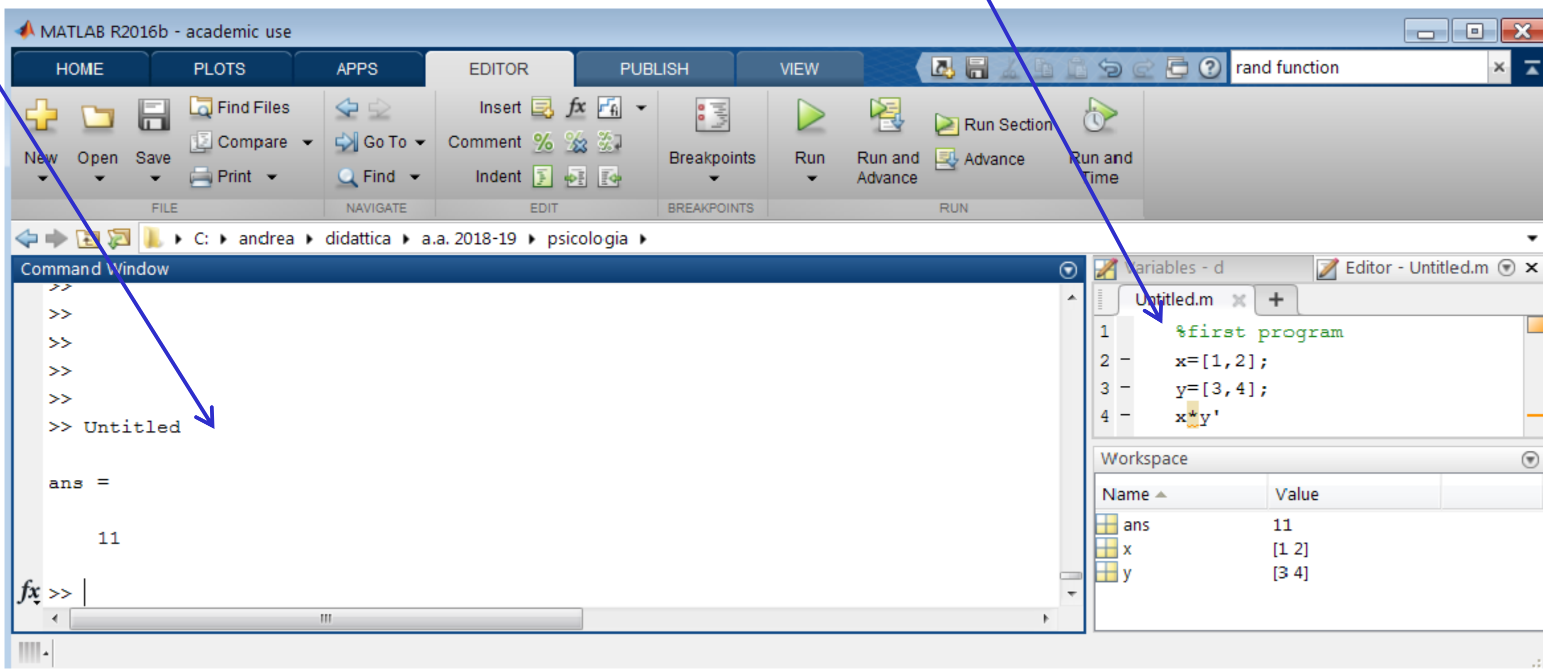
Add to the variable *test* a further field storing the total points obtained by each subject.

Chapter 4– Start Programming

From now on we acquire the possibility of writing sequences of (structurated) commands in a friendly and immediate way.

To do that, MATLAB provides a text editor accessible from the EDITOR label.

You write your sequence of commands on the right panel and run them on the left panel simply typing the name of the related file (hereafter Untitled.m, saved automatically in the working directory. The name can obviously be changed.)



Chapter 4– Start Programming
Control flow statements

Cycles and Conditionals: *if*

Syntax
if condition
statement1
else
statement2
end

Multiple conditions
if condition1
statement1
elseif condition2
statement2
elseif
…
else
statementn
end

Semantic

If condition is true then statement1 is performed and go to end, else statement2 is performed.

Multiple conditions

If condition is true then
perform statement1 and **go to end**,
else if condition2 is true then
perform statement2 and **go to end**,
else …
else statementn is performed.

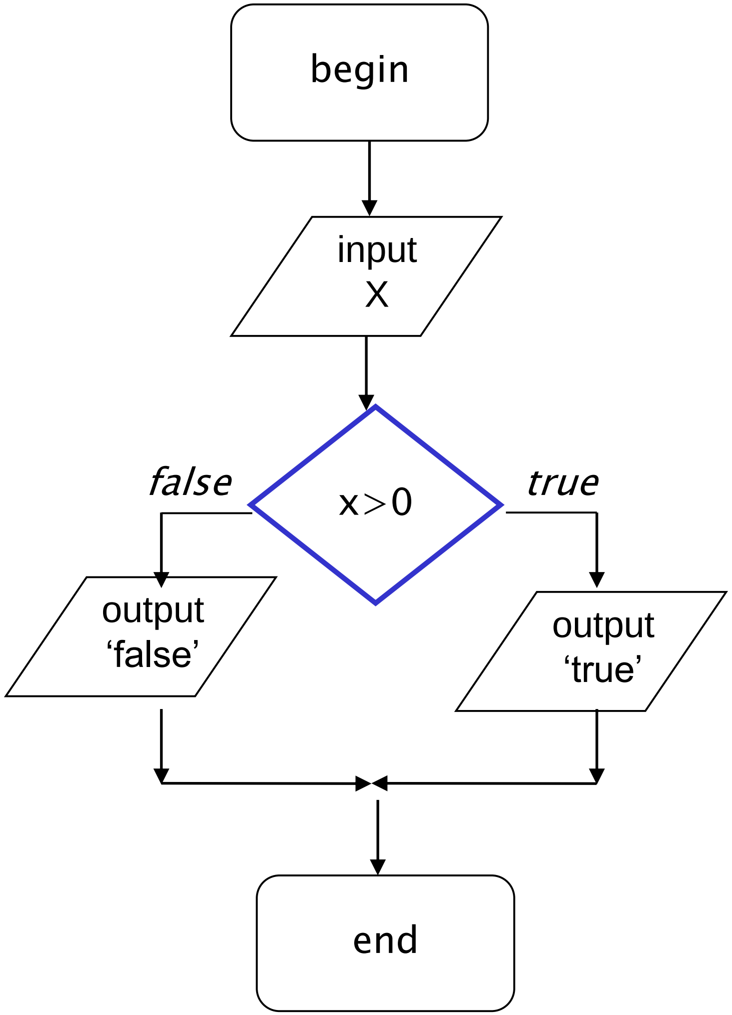
An alternative to the **if – else** form is the **switch – case** form that sometimes leads to more readable code.

Chapter 4– Start Programming
Control flow statements

Cycles and Conditionals: *if*

```
Test_if.m
x=input('Insert a number greater than zero:')
if x>0
   disp('true')
else
  disp('false')
end;

>> Test_if
Insert a number greater than zero:  45
>>true
```

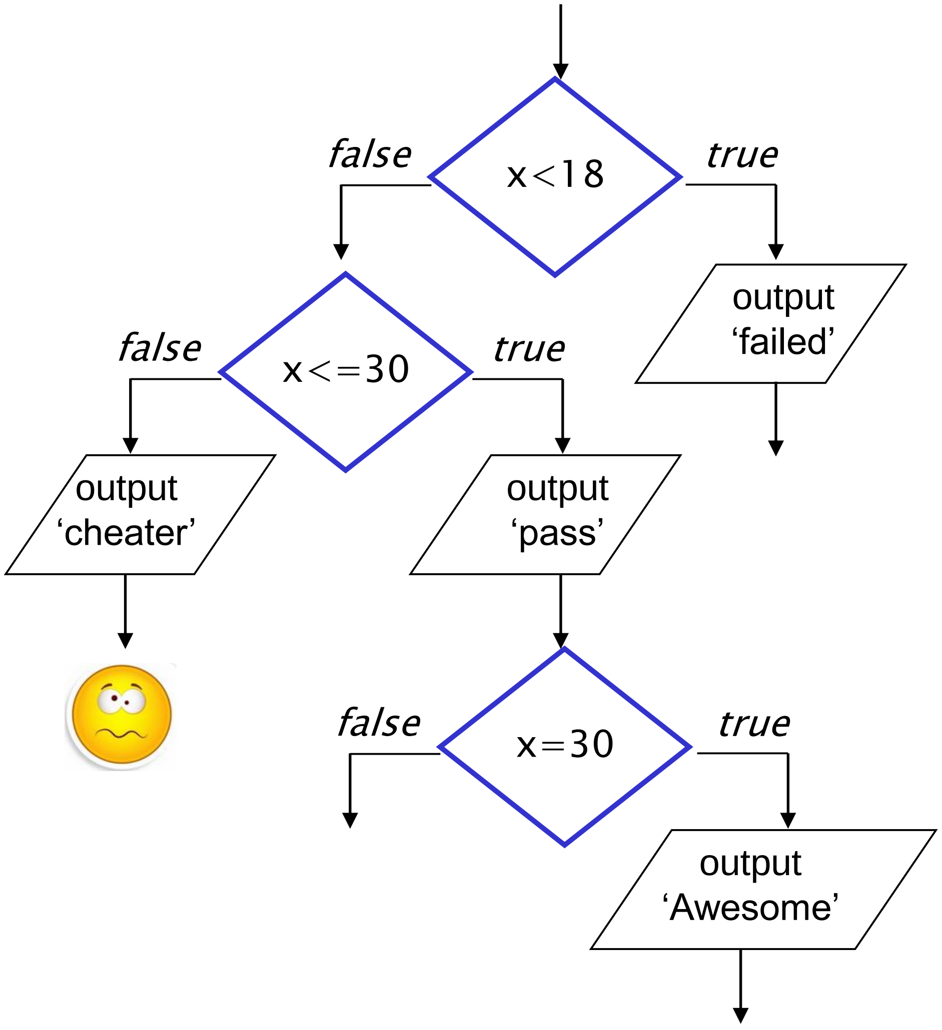
```
Test_mult_if.m
x=input('Insert test result [0-30]:');
if x<18
   disp('try again')
elseif x<30
  disp('Good result: you pass!')
elseif x=30
  disp('Awesome!!!')
else
  disp('you cheater')
end;

>> Test_mult_if
Insert test result [0-30]: 28
>> Good result: you pass!
```



Chapter 4– Start Programming
Control flow statements

Cycles and Conditionals: *if*

```
Test_nested_if.m

x = input('Insert test result [0-30]:','s');
if str2num(x)<18
   disp('you failed')
elseif str2num(x)<=30
  disp('Good result: you pass!')
     if strcmp(x,'30')
       disp('Awesome!!!')
     end;
else
  disp('you cheater')
end;

>> Test_nested_if
Insert a number greater than zero:  30
>>Good result:you pass!
>>Awesome!!!
```



Chapter 4– Start Programming
Control flow statements

For Loops: it fulfills the need of repeating a block of statements a number of times

Syntax

for var in *list_of_values*
statement
end

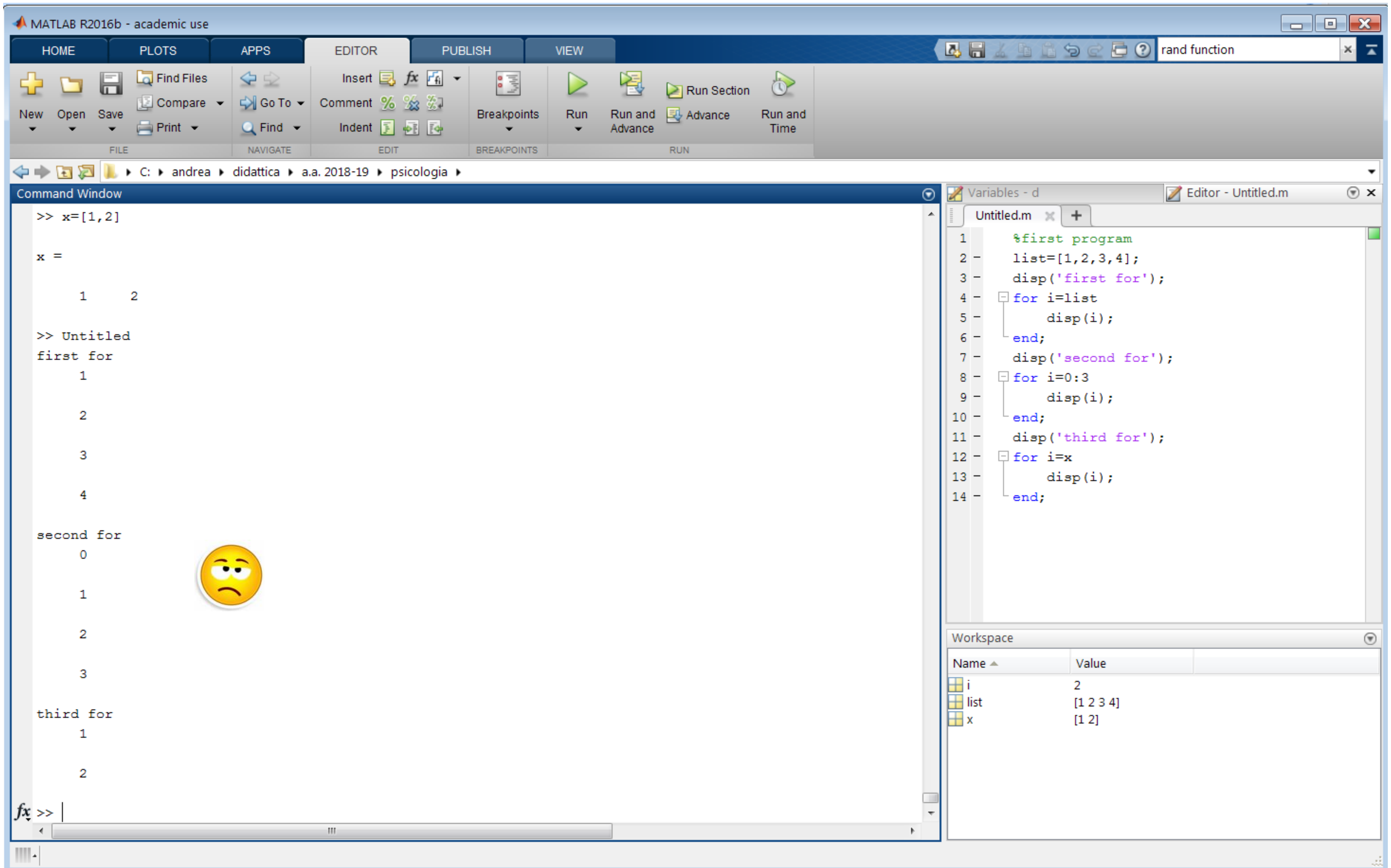
for var in *start:step:stop*
statement
end

Semantic

The variable *var* takes all the values in *list_of_values* and **for each of them** *statement* is performed

The variable *var* takes all the **integer** values from *start* to *stop* each time increasing/decreasing of *step* and **for each of them** *statement* is performed

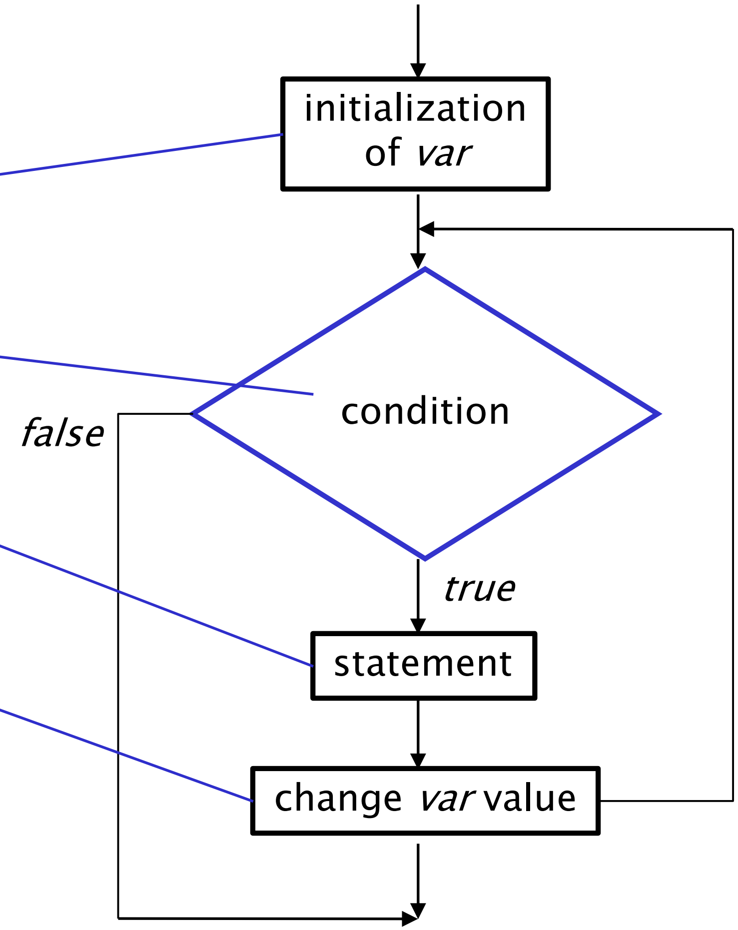
Chapter 4– Start Programming
Control flow statements



Chapter 4– Start Programming Control flow statements

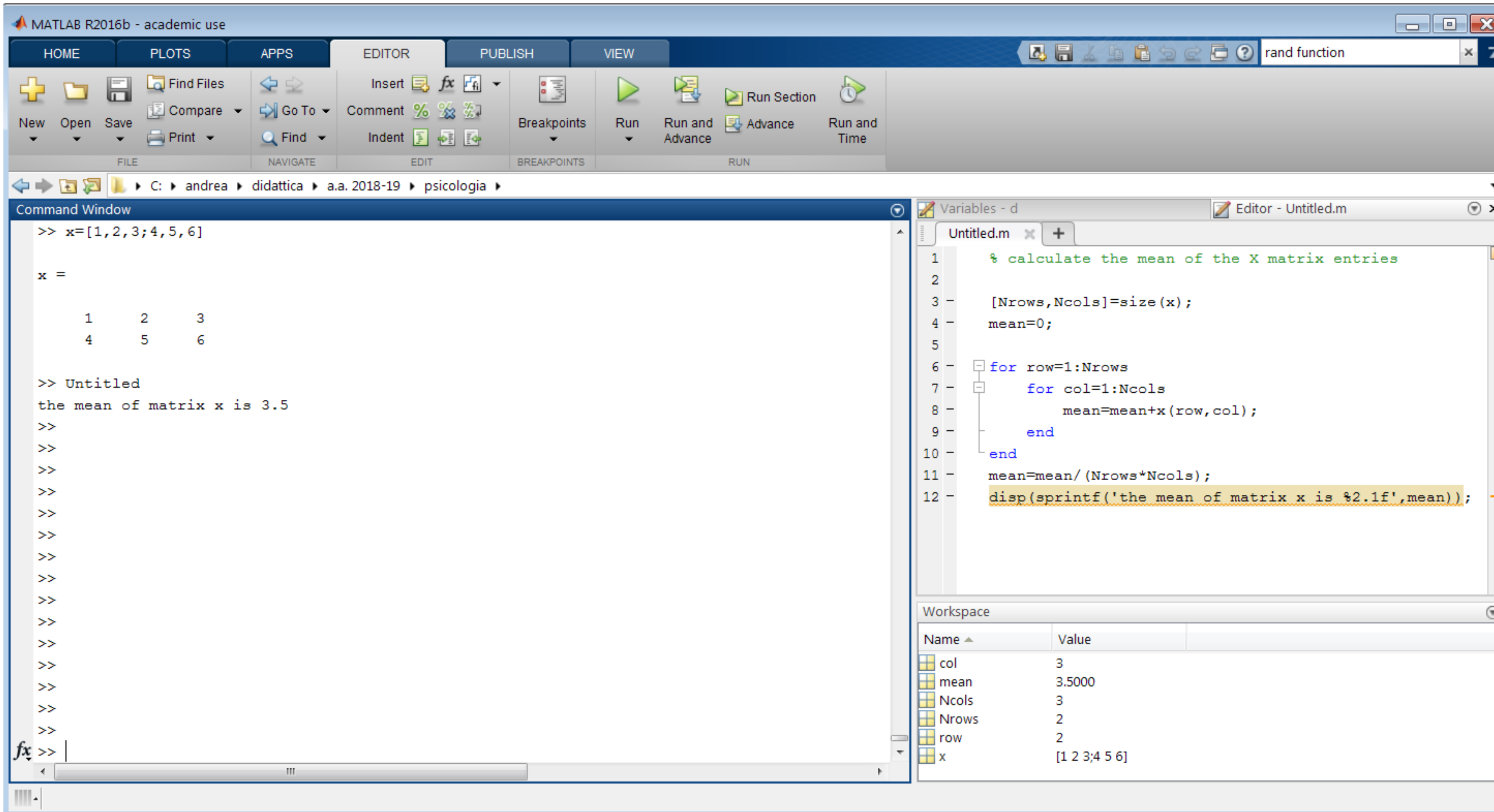
For Loops: it fulfills the need of repeating a block of statements a number of times

```
Test_for.m

for var=3:2:10
   disp('the value of var is %d',var)
end;

>> Test_for
       3
       5
       7
       9
```



Chapter 4– Start Programming
Control flow statements

EXTREMELY USEFUL EXAMPLE (NESTED *for*)



Chapter 4– Start Programming
Control flow statements

While Loops: it repeats a block of statements *while* a condition is true (so indefinitely many times)

The while and the for loops can be used equivalently; they are only *more adequate* to different situations.

```
% mean of a vector x

Nelem=langth(x);
mean=0;
Index=1;
while  index<=Nelem
  mean=mean+x(index);
  index=index+1;
end
mean=mean/Nelem;
disp(sprintf('the mean of x is %2.1f',mean));
```

```
%create a menu
ans=‘ ';
ansvect=char('S','V','E');
while ~(ismember(ans,ansvect))
    disp('Do your own choice:');
    disp('S: start experiment');
    disp('V: visualize last trial''s result');
    disp('E: exit');
    ans=input('Do your own choice','s');
end;
disp('good choice');
```

Listing 4.9 shows an interesting and simple application of while loop for adaptive procedures

Chapter 4– Start Programming
Control flow statements

break: <u>break command forces the exit from a loop, sometimes it is very useful</u>

the following is a quite artificial example

```
% mean of a vector x using break
Nelem=length(x); mean=0; index=1;
while 1            % neverending loop
    mean=mean+x(index);
    index=index+1;
    if index<Nelem
       break;
    end;
end;
disp(sprintf('the x mean is %2.2f',mean));
```

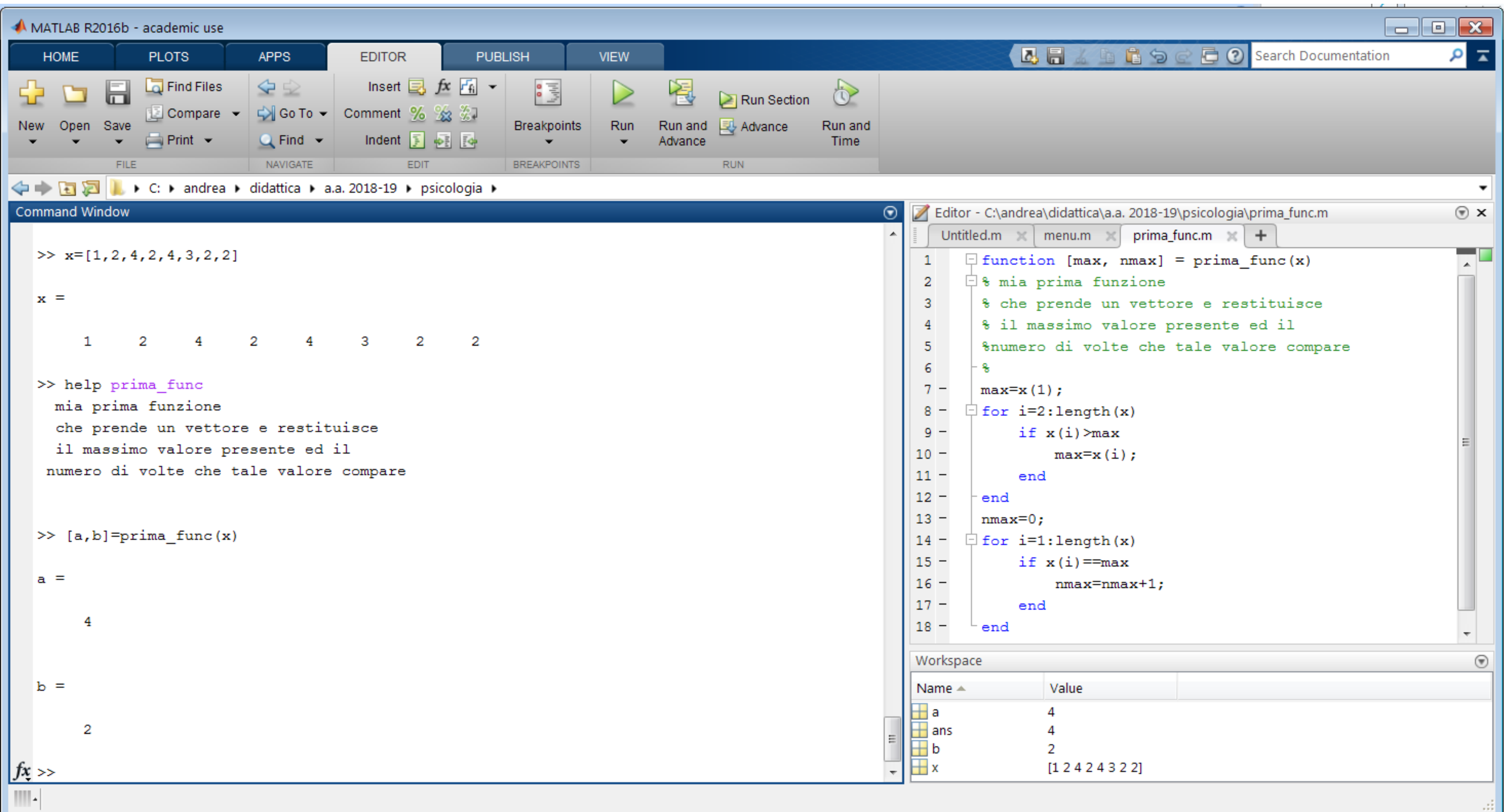
Read the paragraph *Try-Catch*

Skip the paragraph *Loop Versus Matrices and if Versus Logicals*

Chapter 4– Start Programming Functions

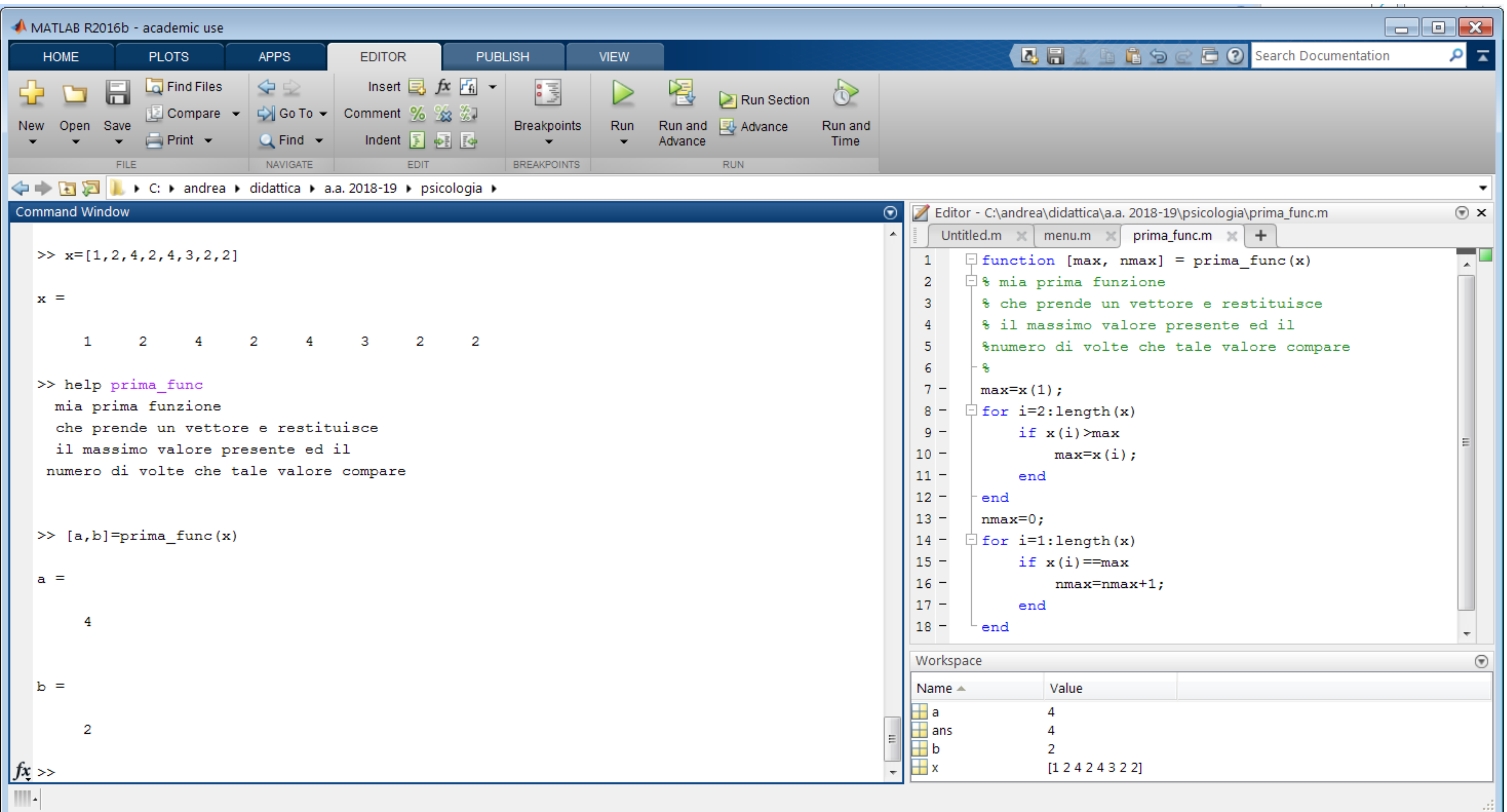
Scripts that receive INPUTS and return results as OUTPUTS are called *functions*. Examples of “built-in” functions in MATLAB are *sin, sum, length* …

Functions scripts start with the reserved word *function* and the .m file has to **match** the name of the function:



Chapter 4– Start Programming Functions

> *ATTENTION*: the input and output variables are dummies and serve only to point out how the function communicates with the workspace



Chapter 4- Start Programming Functions

Scope of variables

visibility or accessibility of a variable from different parts of the program

When a function is called, the variables defined inside it are created (if already present the old ones are frozen) and lasts till the end of the function. Those variables are called **LOCAL VARIABLES**.

GLOBAL VARIABLES: usually written in capital letters, they are defined in the workspace and the are accessible from all the procedures.

PERSISTENT VARIABLES: they can be defined only inside functions and live in the space where they are created. They persist between successive calls of the function.

Chapter 4– Start Programming
Functions

Scope of variables: examples

LOCAL VARIABLES	GLOBAL VARIABLES	PERSISTENT VARIABLES

>>x=2
function test_loc
x=0
 	
>>test_loc
x = 0
>>x
x = 2

	>>global MYVAR; MYVAR=0
	
	Test_glob.m
	disp(MYVAR);
	MYVAR=MYVAR+1;
	fprintf(' ancora %d \n',MYVAR);
	
	>>test_glob
	MYVAR = 0
	MYVAR = 1
	>>MYVAR
	MYVAR = 1
	>>test_glob
	MYVAR=1
	MYVAR=2
		function [z] = test_pers()
		persistent y;
		if isempty(y)
		y=0;
		end
		y=y+1;
		z=y;
		
		>>test_pers
		Ans = 1
		>>y
		Undefined y variable
		>>test_pers
		Ans=2

Chapter 4- Start Programming Functions

Change the number of inputs and outputs: if we need to change the number the inputs of a function, we have to use the *varargin* (variable arguments in) and *nargin* (number of arguments in) variables.

If we need to do the same with the outputs of a function, we similarly have to use the *varargout* (variable arguments out) and *nargout* (number of arguments out) commands.

Varargin and varargout are cells variables, i.e., arrays of input variables whose access to the *i-th* element is *varargin{i}*, and *varargout{i}* (see their use in the example below). *Nargin* and *nargout* are integers.

```
function [mea, varargout] = test( x,varargin )
fprintf('Number of input: %d\n',nargin);
fprintf('Number of output: %d\n',nargout);
Nelem=length(x);
Selem=sum(x);
for i=1:nargin-1
   Nelem=Nelem+length(varargin{i});
   Selem=Selem+sum(varargin{i});
end
mea=Nelem;
varargout{1}=Selem/Nelem;
```

```
>>x1=[2,3,1,2];x2=[3,9];x3=8;
>> [y]=test(x1)
Number of input: 1
Number of output: 1
y = 4

>> [y,z]=test(x1,x2,x3)
Number of input: 3
Number of output: 2
Y=7
z=4
```

Chapter 4– Start Programming Functions

Change the number of inputs and outputs – additional exercises

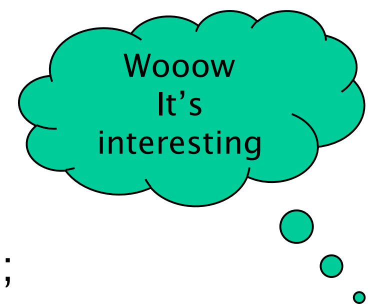
1. compute the maximum and minimum (2 different outputs) of a sequence of numbers (or vectors of numbers) passed as arguments of a function
2. propose a Menu that, according to the choices Max or Min, applies the previous function to a sequence of numbers provided as input by the user. In addition the menu proposes a third choice to play "*paper, scissors or stone*" in one player mode (the game has to be fair, so use the random function), keeping track of the best players in the Hall of Fame (hint: use the structure HOF global variable) to be displayed on demand.

Chapter 4– Start Programming
More on Data import/Export

Script Examples

Handling files (creating, saving, updating them) is not an easy task: the general philosophy that lies behind these actions involves the use of an **integer pointer** variable (say handler), i.e., a number that keeps track of the last examined symbol of the file.

The last symbol of a file is called ***eof*** (End of File).



To interact with a file, it has to be

– opened (the pointer is set in its first position);

– read (the pointer increases its value by one or more positions);

– updated (a symbol can be changed or new symbols can be added at its end);

– closed (the pointer variable is trashed) after its use.

Each action obviously has its own command to be performed:

Chapter 4– Start Programming
More on Data import/Export

Script Examples (see pg.91, listing 4.15)

```
function displayfile(filename)

x=fopen(filename);

if x==-1
   fprintf('Unable to open %s \n',filename);
else

  while ~(feof(x))

      line=fgetl(x);

      disp(line);
  end

  fclose(x);

end
```

fopen(filename) open the file *filename* and set the handler *x* to its first position.

The command *fopen* returns -1 if the file is not found or problems in its opening occurred.

feof(x) checks if *x* reached the last position of the file,i.e., the *eof* position, and returns the related boolean.

fgetl(x) read the file from the handler till the end of the line.

fclose(x) close the file by unsetting its handler *x.*

Chapter 4- Start Programming
More on Data import/Export

Script Examples

The command fopen(filename,option) presents different behaviors according to the option:

-‘r+’ : the file is opened in read-only mode. No modifications are allowed:

-‘w’ : the file is opened in read\write mode. It allows modifications and if it does not exist, it is created;

-‘a’ : the file is opened in append mode, i.e. it can be modified and the pointer is set to the eof position. Again if the file does not exist, it is created.

The other reading/writing commands may have different options too, that will be used if needed.

Listing 4.16 shows an interesting example of file creation related to an experiment about *iconic memory.*

Chapter 4– Start Programming
Guidelines for a Good Programming Style

Writing code: some hints on how to write a good code:

- modularity is a winning strategy (small and well designed functions are useful and easy to be reused). Define clearly INputs and OUTputs.

- check the variables life and values prompting them as much as possible. Do donkey tests inserting strange and unexpected inputs. Communicate errors to the users;

- use indentation!

-comment your script and use instructions on how functions work.

- use meaningful variables, also with long names if necessary.

- use the debug functionality. MATLAB has it by default and can be activated using breakpoints (see *Debug* section of the book).

LAST BUT NOT LEAST

do the suggested exercises and dirty your hands writing down lines of code.

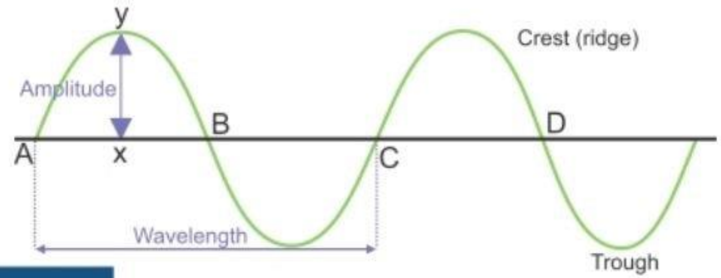
At the end of this chapter you should be able to complete exercises 1.1,1.2,2,3,5, read and understand A Brick of an Experiment,pg.102, and listings 4.18 and 4.19.

Chapter 5–A Better Sound

Generate a Sound

MATLAB provides an easy way to create and manipulate sounds. In the next slides there is a sketch of what it can be done.

Characteristics of SOUND WAVE



Terms related to a wave:

Oscillation	Wavelength	Amplitude	Time period	Frequency
One complete to and fro motion, where one full wave is constituted. From fig, If a wave starts from 'A', it completes full wave at 'C', making one oscillation.	Length of a wave along x-axis, represented by 'λ' (lambda). From fig, AC is the wave length. It is measured in Angstrom unit (A). **1 A = 10^{-10}m**	The maximum displacement of a wave on either side of its mean position. From fig, XY is the amplitude of the wave.	The time taken by a wave to complete one oscillation. It is denoted by **'T'.**	The number of oscillations made by wave in one second. **It is denoted by 'n' or 'f'. Its unit is hertz (Hz).**

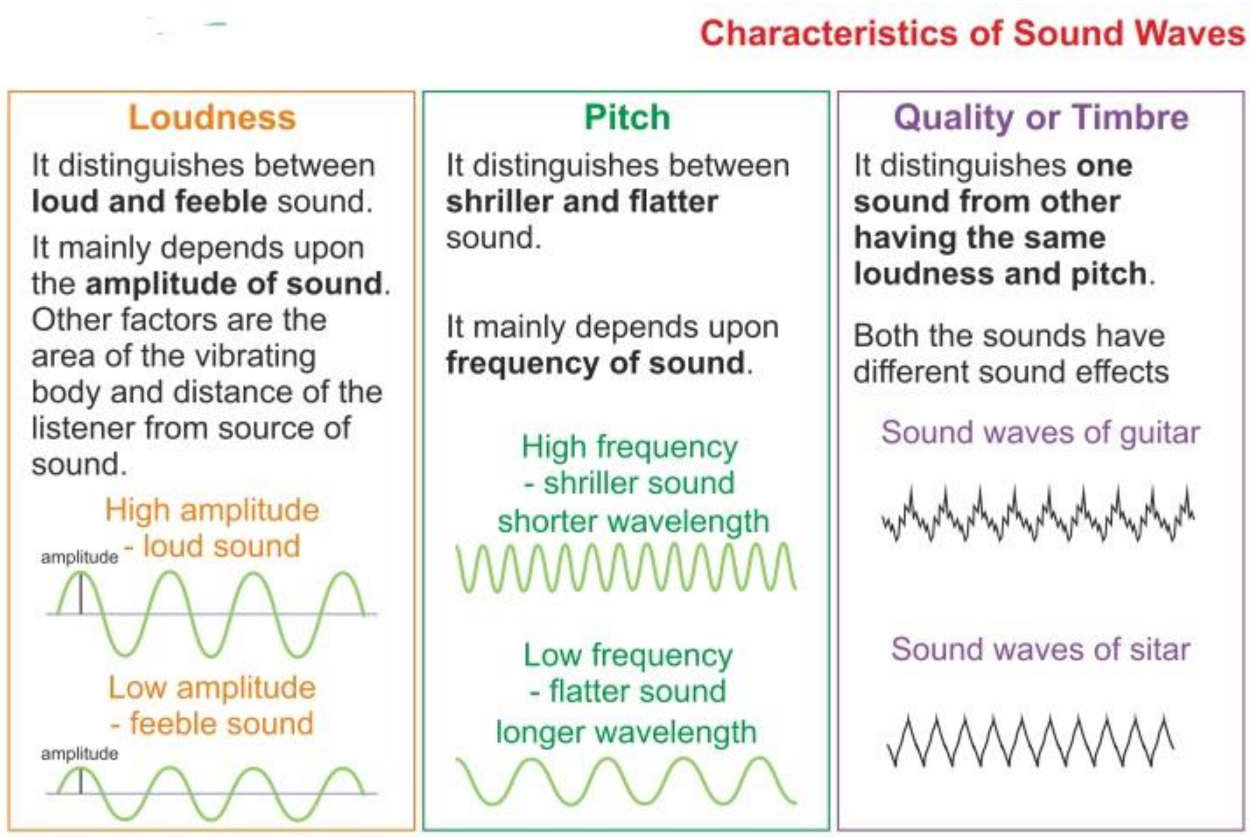
Relation between Time period and Frequency

Time period = 1/Frequency
Frequency = 1/Time period

Chapter 5-A Better Sound

Generate a Sound

MATLAB provides an easy way to create and manipulate sounds. In the next slides there is a sketch of what it can be done.



Chapter 5-A Better Sound

Generate a Sound

MATLAB basically uses the *sound* command to generate sounds:

sound(tone_values,frequence) generates a sound using the values of the array *tone_values*, and playing *frequence* of them each second.

N.B. in order to avoid sound distorsions, the values of *tone_values* have to be normalized in the range [-1,1].

Generate a random sound	Generate a sound with given frequency

sr=44100; % samples per second, in Hz
d=1; % time duration of the sound
noise=rand(1,sr*d); % generates a vector of length sr*d with random elements in the interval [0,1]
noise=noise*2-1; % see the N.B. above
sound(noise, sr); % play the values of noise with frequency 44100 each second
	sr=44100; d=1;
	f=1000; % frequency of the sound
	t=linspace(0,d,sr*d); % check what is generated
	angle=2*pi*sr*d; % a sequence of sr*d angles, from 0 to the length of the needed wave, is generated
	tone=sin(angle); % computation of the tones
	sound(tone, sr);

	% linspace can also be implemented as
	t=[];
	for i=1:sr*d
	    t=[t,i];
	end
	t=t/sr*d;

Chapter 5-A Better Sound

Generate a Sound

errata corrige pgg.108-109:

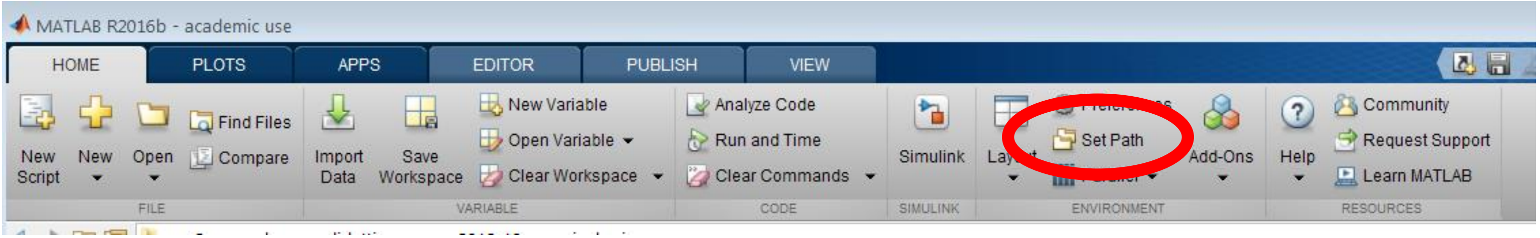
to record the created sounds in a sound file format, usually wave, one can use the *psychwavwrite(tone,sr,'my first wave.wav')*, function that is defined in the psychtoolbox (search for file and instructions in the folder Psychtoolbox/Psychsound).

To add a toolbox to your default MATLAB installation

- download it;

- unzip it in a folder you like (folder suggested name: name_of_the_toolbox);

- add the folder to MATLAB path (click on *Set Path*) by the *Add with Subfolders* option



Chapter 5-A Better Sound

Generate a Sound

a generic sound is the composition of various *harmonics* (single waves) having different frequences and amplitudes. First we learn how to combine three waves with different frequences, both having a common base frequence of 250Hz, and not having (the effect is the same as pro or noob horseriding).

Harmonic 250 Hz sound (sawtooth wave)	Inharmonic sound – different frequencies composition

sr=44100; f=250; d=1;
t=linspace(0,d,sr*d);
 	
first_wave=sin(2*pi*f*t);
second_wave=sin(2*pi*(2*f)*t);
third_wave=sin(2*pi*(3*f)*t);
 	
harmonic=first_wave+second_wave+third_wave;
harmonic=harmonic/max(abs(harmonic));
sound(harmonic,sr);
subplot(2,2,1); plot(first_wave(1:500));
subplot(2,2,2); plot(second_wave(1:500));
subplot(2,2,3); plot(third_wave(1:500));
subplot(2,2,4); plot(harmonic(1:500));
	sr=44100; d=1;
	
	f1=200; f2=250; f3=380;
	
	t=linspace(0,d,sr*d);
	first_wave=sin(2*pi*f1*t);
	second_wave=sin(2*pi*f2*t);
	third_wave=sin(2*pi*f3*t);
	inharmonic=first_wave+second_wave+third_wave;
	inharmonic=inharmonic/max(abs(inharmonic));
	sound(inharmonic,sr);
	subplot(2,2,1); plot(first_wave(1:500));
	subplot(2,2,2); plot(second_wave(1:500));
	subplot(2,2,3); plot(third_wave(1:500));
	subplot(2,2,4); plot(inharmonic(1:500));

Chapter 5-A Better Sound

Generate a Sound

In order to obtain a better *sawthoot wave*, base for most of the synthesized instruments' sounds, we have to act also on the waves' amplitudes, usually by halving it time after time. The most waves are used, the most the final wave resembles the sawtooth one.

Acting on amplitudes	Multiple Sounds

sr=44100; f=250; d=1;
t=linspace(0,d,sr*d);
first_wave=1*sin(2*pi*f*t);
second_wave=0.5*sin(2*pi*(2*f)*t);
third_wave=0.25*sin(2*pi*(3*f)*t);
harmonic=first_wave+second_wave+third_wave;
harmonic=harmonic/max(abs(harmonic));
sound(harmonic,sr);
	sr=44100; d=0.5;
	f_do=261.6;
	f_re=293.6;
	f_mi=329.6;
	t=linspace(0,d,sr*d);
	do=sin(2*pi*f_do*t);
	re=sin(2*pi*f_re*t);
	mi=sin(2*pi*f_mi*t);
	silence=zeros(1,sr*d);
	sound([do, re, mi, silence, do],sr);

The last example shows how to generate a small melody of a couple of seconds by simply concatenating 5 different sounds

Chapter 5-A Better Sound

The remaining part of the Chapter (pgg.113-125) goes deep into the sound creation and manipulation, and it is skipped here.

Suggested exercises: pg.125, from 1 to 6

Chapter 6 – Create and Process Images

Images Basics

an image is represented as an integer valued matrix, each element representing a colored pixel. The admissible values of each pixel are:

– *grey intensities*: represented as 8 bits numbers raging from 0 (black) to 255 (white);

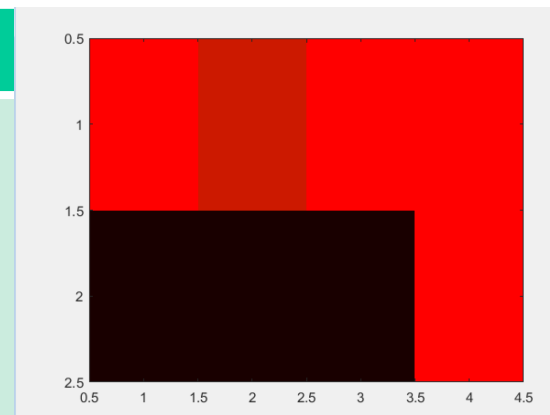
– RGB triplets: triplets of intensity values of the colors Red Green and Blue, raging from 0 to 255 each and yelding to 2^{24} different colors (True Color);

–indexing color: a number chosen in a 64 colors palette table and corresponding to an assigned triplet of RGB nuances.

MATLAB uses indexing images by default

Showing palette table	Changing palette table

colormap %shows palette table
MMap=colormap;
MMap(3:6,:)
Ans =
0.2123 0.2138 0.6270
0.2081 0.2386 0.6771
0.1959 0.2645 0.7279
	MMap=[1,0,0;0.8,0.1,0;0.1,0,0]
	colormap(Mmap);
	% default colormap changes into MMap
	img=[1 2 1 1;3 3 3 1];
	image(img) % img is shown



Chapter 6 – Create and Process Images

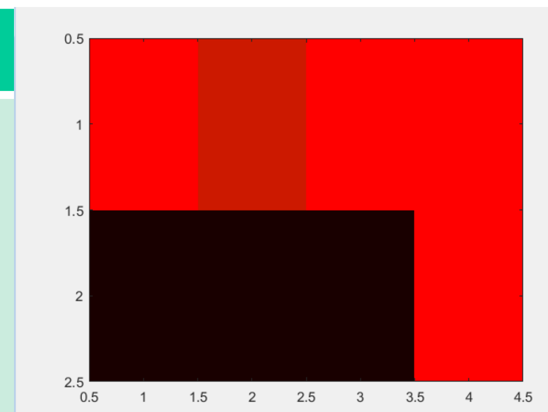
Images Basics

ATTENTION: the default palette table is restored once the Fig. environment is closed.

a nice and useful way to change palette table:

>>colormapeditor 

Showing palette table	Changing palette table	

colormap %shows palette table
MMap=colormap;
MMap(3:6,:)
Ans =
0.2123 0.2138 0.6270
0.2081 0.2386 0.6771
0.1959 0.2645 0.7279
	MMap=[1,0,0;0.8,0.1,0;0.1,0,0]
	colormap(Mmap);
	% default colormap changes into MMap
	img=[1 2 1 1;3 3 3 1];
	image(img) % img is shown
		

Chapter 6 – Create and Process Images

Importing and Exporting Images

an image can be imported from outside into a variable matrix using the command

variable=imread('filename';'file type');

as well it can be overwritten/created using the command

imwrite(variable;'filename';'file type');

ATTENTION: according to the file format (tiff, png, bmp,jpg,gif…), variable has different formats. See manual for references.

Importing an image	Comments
% choose a small colored image, say icon.bmp A=imread('icon.png','png') % imported matrix is displayed in numeric format image(A) % image is displayed imwrite(A,'icon2.png','png') % icon2 is created in the default folder	*In our example, the obtained matrix has dimension 128*128*3 since each pixel is expressed in RGB values raging from 0 to 255 (8 bits representation).* *No further infos are present so using the command* *[A,B]=imread('icon.png,'png)* *B turns out to be void.* *Images may have a proper palette table as additional info, that, in case, is imported into B*

Chapter 6 – Create and Process Images

Display images

there are two main functions to display images after importing with *imread* command:

–image(A);

–imshow(A,colormapofA); %colormapofA is the map color obtained with *imread*

To obtain a grayscale (100) color map use the command *colormap(gray(100))*

For three dimensional image data the colormap is ignored

Trick: the command axis off avoid displying the axis

Exercise:

create a random 128 x 128 image and display it changing the colormap

Chapter 6 – Create and Process Images

Intensity transformation:

an image can be regarded as an integer matrix and as so, it can be manipulated:

as an example, we can enjoy increasing/decreasing its brightness by adding/subtracting to all of its entries the same value, here on128.

Intensity transformation	Comments
A=imread(‘mandrill.jpg’,’jpg’); Alight=floor(min(A+128,255)); % shift hight the color components of mandrill Adark=floor(max(A-128,0)); % shift low the color components of mandrill A3=256-A; % invert the intensities of mandrill subplot(1,4,1);image(A); axis off subplot(1,4,2);image(A1); axis off subplot(1,4,3);image(A2); axis off subplot(1,4,4);image(A3); axis off %plot everything	*The floor function round a number to its maximum lower integer.* *The functions min and max allow not to exceed the 0-255 values range.* 

Chapter 6 – Create and Process Images

Intensity transformation:

to change a rgb image into a grayscale one use the command *rgb2gray()*

now it is even more evident the action of the brightness filtering

Changing into grayscale	Comments
A=imread('mandrill.jpg','jpg'); *C=rgb2gray(A)* *% mandrill is grayscaled* *colormap(gray(256));* *image(C);* *% plot the gray scaled mandrill* *image(C')* *% rotate mandrill*	*Again considering mandrill image as an integer matrix allows us to perform matematical operations on it.*

Chapter 6 – Create and Process Images

Windowing:

Enhance some parts of an image by multiplying it with a window of the same size whose entires are usually in the range [0,1]. A first example selects the central part of an image and the second enhances it with a gaussian window (see listing 6.2)

Create a selecting window	Comments

A=imread('mandrill.jpg','jpg');
A=rgb2gray(A)
% mandrill is grayscaled
window=zeros(A);
centX=size(A,1)/2
centY=size(A,2)/2; %compute the center of A
winsize=50 % size of the window
window([-winsize:winsize]+centX, [-winsize:winsize]+centY)=1
% the center of the window is set to 1
newimage=A.*window;
% the windowed image is created
imagesc(newimge);
	A window of the same size of an input image that cuts its central 50x50 squared part is created.
	It is applied via standard multiplication to the input image (here mandrill.jpg)

Chapter 6 – Create and Process Images

Neighborhood processing (read)

The Edges of the Image (read)

Advanced Image Processing (read)

Chapter 6 – Create and Process Images

Creating Images by Computation:

let us now approach the design of simple images. This argument will be treated in the Psychtoolbox chapter. The following example shows hot to create a line, a polyline figure and a circle.

Create different figures with two simple commands	Comments

% create a polyline with three points
line([-1,2,4],[-2,0,3])
% create a red triangle
fill([-1,2,4],[-2,0,3],'r')
% create a circle as a closed polyline
Npoints=30;
x=[1:Npoints]./Npoints*2*pi;
radius=3;
fill(radius*sin(x)+2,radius*cos(x)+1,'r')
	A series of elements is depicted.
	A red circle whose center is in the point (2,1) and the redius equal to 3 is created.

Exercises 1 and 2 are suggested

Chapter 7 – Data Analysis

Descriptive Statistics

Measures of Central Tendency

mean(v), mode(v) and median(v)

geomean(v), harmean(v) and trimmean(v,percent)

Measures of dispersion

max(v), min(v), std(v), var(v), …

for additional measures see the Statistics toolbox

Bivariate and Multivariate Descriptive Statistics

Covariance

Simple and Multiple Linar Regression

Generalized Linear Model

All the functions have a standard syntax and are easy to use when needed

Chapter 7 – Data Analysis

Inferential Statistics

Parametric Statistics

… t-Test (see example below)*…*

ANOVA

Nonparametric Statistics

Test if 20 random numbers' mean is different from 0	Comments
[H p CI stats] = ttest(rand(20,1)) H = 1 % the null hypothesis CAN be rejected p = % probability of finding these results by random chance % very low 4.6959e-08 CI = % confidence interval of the mean 0.4876 0.7964 stats = % parameters of the t-test struct with fields: tstat: 8.7026 df: 19 sd: 0.3299	*The t-Test is performed by the ttest(v) function that tests if the mean of a vector of values is different from 0 with a significance level of 0.05. If a value different from 0 is required, it is the second argument input.* *The ttest(v) function also accepts left and right parameter to specify the direction of the tail test (test >0 or <0 only).*

Chapter 8 – The Charm of Graphical User Interface

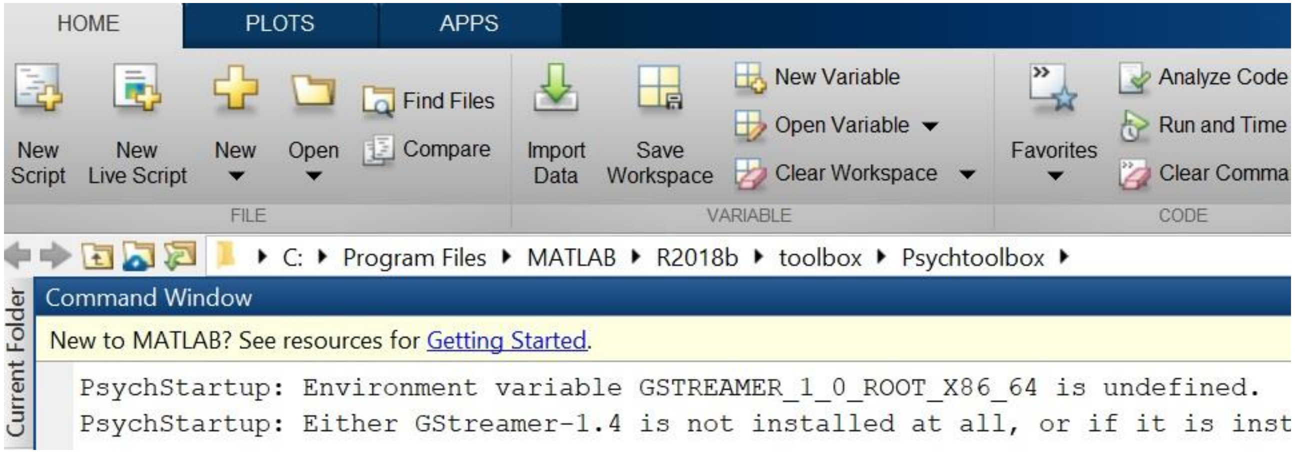
In this chapter it is introduced a firendly way to allow the user to interact with a program we have created.

This part uses notions from the paradigm of Object Oriented programming and it overcames the aims of the course.

We will introduce some of the functionalities here skipped in the next chapter using some functions of the Psychtoolbox.

PsychToolbox installation hints

1. Go to psychtoolbox.org. Download and install the version of PsychToolbox compatible with your PC Operating System.
2. Open Matlab and set the PsycToolbox folder as working folder.



3. Run >>SetupPsychtoolbox
4. Answer ‘no’ and then ‘yes’ to the Matlab requests. Finally press enter two times.

Chapter 9 – Psychtoolbox: Video

The Screen Function

this is the core function of the toolbox and it is mainly used to manage graphical functions and parameters as draw geometrical shapes, import figures, get info about the HW and SW characteristics and synchronize all the stimuli.

Its general call is *Screen('SubFunctionName',parameter1,parameter2,...)*

whose help file is *Screen('SubFunctionName?')*

the following SubFunctions provide info about the HW and SW:

Version (version of PTB), *Computer*, *Screen* (the screens connected to the PC), *FrameRate* ...

Starting with Screen function	Comments
>>Screen('FillRect?') *Ans =* *Screen('FillRect', windowPtr [,color] [,rect])* *>>Screen (Computer)*	Ask for help to the *fillrect* function The parameter [...] are considered as optional The charateristics of the computer are displayed in a *Struct* variable form

Chapter 9 – Psychtoolbox: Video

The Screen Function

the use of *try … catch … end* is here extremely useful and it allows to bypass loops of errors with a timeout or overload detect procedure.

Example	Comments
A=imread('mandrill.jpg','jpg'); C=zeros(size(A,1),size(A,2)); *try* B=A.*C image(B) *catch* disp('Error in something'); *end*	A and C are same-size, different-type matrices, so the .* operator provide an error. Instead of showing it, 'Error in something' is displayed.

Chapter 9 – Psychtoolbox: Video

How to use Screen to Draw Figures

the main feature of Screen in to present figures or drawings with the maximal timing accuracy.

Three steps are needed: *open* a figure, *draw/modify* it and *close* it.

Opening the Window

To open a figure one must use '*OpenWindow*' SubFunction.

Its first parameter is the screen where we want to disply the figure (in case of multiscreens); the default parameter is 0. After a color RGB triplet is optional, and then the area we want to set as window, to draw inside. If no area is specified, then the whole screen area is considered.

The function returns a pointer to the screen and the screen coordinates in pixels (a 4-tuple [0,0,x,y] where (0,0) is the top-left corner, (x,y) is the bottom-right corner of the screen. Other options can be found in the on-line manual.

Some settings are often needed in order to obtain the full functionality of the OpenWindow Subfunction

Chapter 9 – Psychtoolbox: Video

Opening the Window

Example	Comments
[myscreen, rect]=Screen('OpenWindow',0,[0,255,0]);	Open a window of the same size as the screen, and make it green. Myscreen is a pointer to the screen, while rect is a 4-tuple with top-left pixel and the bottom-right pixel coordinates.
Myrect=[10,20,150,250]; *Screen('OpenWindow',0,[],Myrect);*	Open a new rectangular window whose top-left pixel and the bottom-right pixel coordinates are (10,20) and (150,250) of the default white color.

Closing

To close the window and destroy the pointer simply write

Screen('CloseAll')

If we open more than one window, then we can destroy a single one, i.e., its pointer, say *pippo*, using

Screen('Close',pippo)

Chapter 9 – Psychtoolbox: Video

Drawing: an Introduction and *Reprise*

The use of Flip command: when one or more figures are drawn, they are saved in the background memory (*backbuffer*) and so not visible.

The Flip subfunction moves the figures from the backbuffer to the foreground memory (*frontbuffer*), and so they become visible.

When Flip is executed, the backbuffer is cleared and the frontbuffer is updated.

Sintax is:

Screen('Flip',windowPtr)

where windowPtr is a pointer to the chosen screen.

A further useful command:

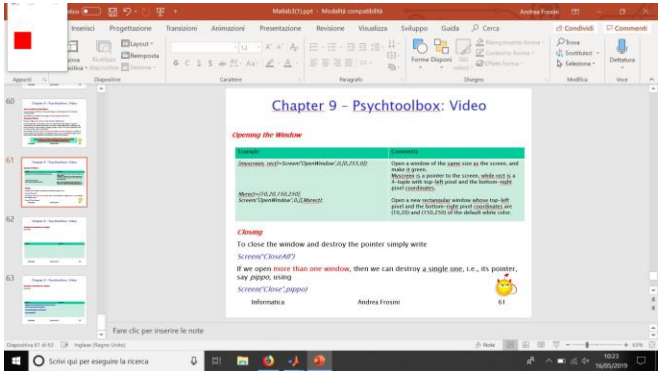
KbWait

that stops the execution of the code until a key-press

Chapter 9 – Psychtoolbox: Video

Drawing: an Introduction and ***Reprise***

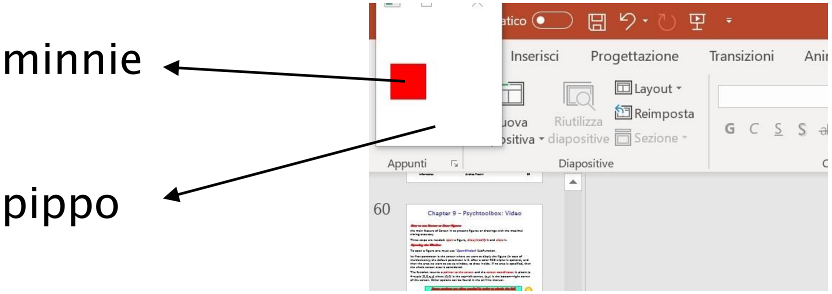
A simple example



Example	Comments
[pippo,pluto]=Screen('OpenWindow',0,[],[10,20,100,200]);	A white small rectangular area is depicted on the top of the screen.
minnie=CenterRect([0,0,50,50],pluto);	A 4-tuple of coordinates is created in minnie that centers the minnie rectangle inside pippo.
Screen('FillRect',pippo,[255,0,0],minnie);	Minnie is red filled and placed inside pippo. Nothing is shown since the rectangle is in background.
Screen('Flip',pippo);	Minnie appears since Flip sets it to foreground.
Screen('Close',pippo);	Pippo is closed and the pointed trashed.

Drawing: an Introduction **and** ***Reprise***

A simple example



Example	Comments
[pippo,pluto]=Screen('OpenWindow',0,[],[10,20,100,200]);	A white small rectangular area is depicted on the top of the screen.
minnie=CenterRect([0,0,50,50],pluto);	A 4-tuple of coordinates is created in minnie that centers the minnie rectangle inside pippo.
Screen('FillRect',pippo,[255,0,0],minnie);	Minnie is red filled and placed inside pippo. Nothing is shown since the rectangle is in background.
Screen('Flip',pippo);	Minnie appears since Flip sets it to foreground.
Screen('Close',pippo);	Pippo is closed and the pointed trashed.

Chapter 9 – Psychtoolbox: Video

Drawing shapes

A simple example. A harder and affordable one is Listing 9.4.

Example	Comments
try *[mywin, mywindim]=Screen('OpenWindow', 0, [0,255,0]);*	 *mywindim* contains the dimensions of the full screen
myrect=[0,0,400,400];	*myrect* contains the dimensions of the rectangle to draw (4-tuple of coordinates)
myplacedrect=CenterRect(myrect,mywindim);	*myplacedrect* contains the coordinates to place *myrect* in the middle of *mywin*
Screen('FillRect',mywin,[255,0,0],myplacedrect); *Screen('Flip',mywin);* *KbWait;* *Screen('CloseAll');* *catch* *Disp('Some errors occurred!');* *End*	A red rectangle is drawn and placed in the backbuffer The rectangle is shown … … until keypressed Finally mywin is closed

Chapter 9 – Psychtoolbox: Video

Drawing shapes

What in the previous page is a way to proceed:

1. set the dimension of the rectangle regardless its coordinates,
2. move it in the desired position (maybe using the functions here on the right).

Function	Description
AdjoinRect	Moves a rect next to another one
AlignRect	Aligns a rect over another one
ArrangeRects	Arranges an array of rects in a pleasant way
CenterRect	Centers a rect within a second one
CenterRectOnPoint	Centers a rect around given x,y coordinates
CenterRectOnPointd	Centers rect around an x,y coordinate pair
ClipRect	Returns the intersection of two rects
ClipRect	Returns the intersection of two rects
InsetRect	Shrinks/expands rect by additive insets
IsEmptyRect	Returns 1 if empty, returns 0 otherwise
IsInRect	Is the point inside a rect?
OffsetRect	Shifts rect vertically and horizontally
RectBottom	Returns index of yBottom entry of a rect
RectCenter	Returns the integer x,y coordinates of center
RectCenterd	Returns the exact x,y coordinates of center
RectOfMatrix	Accept an image as a matrix and returns a PTB rect specifying the bounds
RectHeight	Returns the height of a rect
RectLeft	Returns index of xLeft entry of a rect
RectRight	Returns index of xRight entry of a rect
RectTop	Returns index of yTop entry of a rect
RectWidth	Returns width of a rect
RectSize	Returns the width and the height of a rect
ScaleRect	Scales a rect by multiplicative factors
SetRect	Creates a rect (i.e., a vector) from four input coordinates
SizeOfRect	Accepts a Psychtoolbox rect [left, top, right, bottom] and returns the size [rows columns] of a MATLAB array (i.e. image) just big enough to hold all the pixels
UnionRect	Smallest rect containing two given rects

Drawing shapes

If we need to manage circles, the functions on the right can be used

Sub/Function	Command	Description
DrawLine	Screen('DrawLine', windowPtr [,color], fromH, fromV, toH, toV [,penWidth]);	draws a line
DrawArc	Screen('DrawArc',windowPtr,[color],[rect],startAngle,arcAngle)	draws a circular arc unfilled with color (i.e., a Pac-Man-like figure)
FrameArc	Screen('FrameArc',windowPtr,[color],[rect],startAngle,arcAngle[,penWidth][,penHeight] [,penMode])	as above
FillArc	Screen('FillArc',windowPtr,[color],[rect],startAngle,arcAngle)	as above but filled with color
FillRect	Screen('FillRect', windowPtr [,color] [,rect]);	draws a rectangle filled with color
FrameRect	Screen('FrameRect', windowPtr [,color] [,rect] [,penWidth]);	draws a rectangle unfilled with color
FillOval	Screen('FillOval', windowPtr [,color] [,rect] [,perfectUpToMaxDiameter]);	draws a filled oval
FrameOval	Screen('FrameOval', windowPtr [,color] [,rect] [,penWidth] [,penHeight] [,penMode]);	draws a framed oval
FramePoly	Screen('FramePoly', windowPtr [,color], pointList [,penWidth]);	draws a framed polygon
FillPoly	Screen('FillPoly', windowPtr [,color], pointList [, isConvex]);	draws a filled polygon

Chapter 9 – Psychtoolbox: Video

Batch Processing: Drawing Multiple Figures at Once **(read only)**

Drawing Text:

The sub-function DrawText allows one to draw text on the screen. The sintax is
Screen('DrawText', windowPtr, text [, x] [, y], [, color] [, ...]);

where x and y are the coordinates of the top left corner of the starting text.

The *DrawText* sub-function returns the coordinates (x,y) of the ending point of the inserted *text*.

Example	Format the text
try *pippo=Screen('OpenWindow',0,[0,255,0]);* *MyText='Ciao Mario';* *Screen('DrawText', pippo, MyText,40,50,[255,0,0]);* *Screen('Flip',0);* *KbWait;* *Screen('CloseAll');* *catch* *disp('errore');* *end*	Screen('TextStyle',pippo,n) % set the textstyle of the window pippo % n ranges from 0 to 7 to have normal, bold, italic ... Screen('TextFont',pippo,'Verdana'); % changes the font into Verdana Screen('TextSize',pippo,36) % set the textsize to 36 % all those sub-functions return the previous value of the changed format

Chapter 9 – Psychtoolbox: Video

Drawing Text:

Exercise:

Draw a sequence of four randomly chosen greetings among ‘Ciao’, ‘Hi Hi’, ‘Bonjour’, ‘Hola’ of all red nuances (i.e., colors from [1,0,0] to [255, 0, 0]) in a randomly chosen position of a yellow screen.

Chapter 9 – Psychtoolbox: Video

Drawing Text:

Exercise:

Draw a sequence of four randomly chosen greetings among 'Ciao', 'Hi Hi', 'Bonjour', 'Hola' of all red nuances (i.e., colors from [1,0,0] to [255, 0, 0]) in a randomly chosen position of a white screen.

Example	Comments
try	
[pippo,dim]=Screen('OpenWindow',0,[],[100,100,700,700]);	
cheers=[''Ciao'',''Hi Hi'',''Bonjour'',''Hola''];	An array of ***strings*** is created
Screen('TextFonts',pippo,'Arial');	
Screen('TextSize',pippo,40);	
for i=[0:10:255]	We use the function ***randi( )*** for a quick way to generate random integers.
x=randi(dim(3));	
y=randi(dim(4));	
*MyCheers=**char**(cheers(randi(4))); % WARNING!!!*	Warning: the sub-function DrawText requires an array of char as text input, so we have to change the type of cheers from string to char!
Screen('DrawText',pippo,MyCheers,x,y,[i,0,0]);	
Screen('Flip',pippo);	
pause(0.1);	
end	Remind that the assignment of a text string to a variable can be done using single quotes, i.e., the type will be array of characters, or double quotes i.e., the type will be a single string.
KbWait;	
Screen('CloseAll');	
catch …end	

Chapter 9 – Psychtoolbox: Video

Importing Images

Screen uses the sub-function DrawTexture to show a picture file that is in our HD.

Three steps are needed:

1. Load the image on Matlab, as seen in Chapter 6
2. Create a texture of the picture (texture is a specific way to encode a RGB or gray level image).
3. Show the picture.

Example	Comments
try *A=imread('mandrill.jpg','jpg');* *r=[0,0,size(A)];* *[pippo,dim]=Screen('OpenWindow',0);* *r=CenterRect(r,dim);* *pic=Screen('MakeTexture',pippo,A);* *Screen('DrawTexture',pippo,pic,[],r);* *Screen('Flip',pippo);* *KbWait;* *Screen('CloseAll');* *catch …end*	*Pic* is a pointer to the created texture Pic is inserted inside the rectangle *r* and drawn in the backbuffer. N.B. the texture and the rectangle must have the same dimension.

Chapter 9 – Psychtoolbox: Video

Video Clips

Video clips can be created as a sequence of images showed one after the other with a small difference in position, providing the effect of movement.

They are usually created by loops as in the following example of a disc that moves from left to right on a white screen:

Example	Comments
try *[pippo, dim]=Screen('OpenWindow',0,[255,0,0],[100,100,700,700]);* *discdiam=20;* *disc=[0 0 discdiam discdiam];* *rect=[200 200 400 400];* *disc=AlignRect(disc,rect,'center','left');* *for i=0:180* *Screen('FillRect',pippo,[255,255,255],rect);* *Screen('FillOval',pippo,[0 0 0],[disc(1)+i,disc(2),disc(3)+i,disc(4)]);* *Screen('Flip',pippo);* *pause(0.01);* *end* *Screen('CloseAll');*	Use of **AlignRect**: align the rectangle ***disc*** inside the biggest rectangle ***rect*** centering disc on the y-coordinate and posing on the left the x-coordinate

Chapter 9 – Psychtoolbox: Video

Video Clips

Listing 9.4 can be read and understood.

Drawing Things at the Right Moment **(read)**

Read and realize A Brick for an Experiment, pg. 245.

Exercise:

Draw the picture Mandrill.jpg on a black screen and successively reduce its size view using the Windowing tool (Chapter 6), till full expiring into a full black screen.

Play a single note repeated all over the process and a final different one toghether with the centered big text ‘Bye Bye’.

Chapter 10 – Sound, Keyboard and Mouse

Timing

- WaitSecs(n) halts the run of the program for n seconds.
- GetSecs gets the time between the start of the PC and the GetSecs call. It is extremely useful to take the time (as a subtraction) between two GetSecs calls (i.e., the visualization of a stimulus and the reaction of the subject).

Priority (skip)

Sound Functions

There are some functions to synthesize and play sounds that are extremely useful for psychological experiments.

The main is *PsychportAudio* whose use is similar to that of Screen.

To play a beep of a given frequency f, duration time d and sample ratio sr type

MakeBeep(f,d,sr);

Sound Functions

A quick example

Example	Comments

try
    f=500;
    d=1;
    sr=48000;
    beep=MakeBeep(f,d,sr); % the beep is generated
    %InitializePsychSound;
    pippo=PsychPortAudio('Open', [], [], [], sr, 1);
    PsychPortAudio('FillBuffer',pippo,beep);
    PsychPortAudio('Start',pippo);
    PsychPortAudio('Stop',pippo,d);
catch
    disp('errore');
end
	N.B. the sample ratio of 44100 is not always supported. In case use 48000.
	
	The Open sub-function contains among others, the sound ratio of the sound that will be played and the number of channels, i.e., how many different sounds will be played together.

Chapter 10 – Sound, Keyboard and Mouse

Getting Participants' Inputs: Keyboard and Mouse Functions

Keyboard Response

There are two main classes of keyboard events: keypressed and character oriented.

Only the first ones will be considered, since most representative for psychological experiments. In particular we consider *KbWait()* that waits for user's input, stopping the script execution untill keypressed.

The function KbWait returns both the time before keypressed and the (code of the) key pressed. This code is a 256 boolean array with one only 1 in the character-pressed-code position (see the example in the next slide).

One can switch between code of a keyboard key and its name by means the function *KbName()*.

Example: KbName('c') returns the code 67, and KbName(67) returns the character 'c'.

Chapter 10 – Sound, Keyboard and Mouse

Press any key to proceed

Press the Spacebar to proceed

Press any key to proceed	Press Spacebar to proceed
try *[pippo,rect]=Screen('OpenWindow',0)* *DrawFormattedText(pippo,'PRESS ANY KEY TO_* *PROCEED','center','center');* *Screen('Flip',pippo);* *KbWait,* *Screen('CloseAll');* *catch* *disp('errore');* *End*	*try* *[pippo,rect]=Screen('OpenWindow',0);* *DrawFormattedText(pippo,'PRESS SPACEBAR TO PROCEED','center','center');* *Screen('Flip',pippo);* *spax=KbName('space');* *[tmp,code]=KbWait;* *while code(spax)==0* *[tmp,code]=KbWait;* *end* *Screen('CloseAll');* *catch* *disp('errore');* *end*

Chapter 10 – Sound, Keyboard and Mouse

Press any key to respond

Here the previous two examples are extended asking the subject to produce a y/n output. Listing 10.7 expresses the code: a sequence of text stimuli are presented and required to the subject a y/n response. The sequence of responses are recorded in a boolean vector.

Exercise 'Animals': extend Listing 10.7 by creating the following game: create a 6 words vector, i.e., 3 animals and 3 objects, and a boolean vector of 'right answers'. Then asks the subject to press **a** (animal) or **o** (object) correctly according to ten times randomly presented words.

At the end of the session compute the total score of the subject.

Reaction time detection

Usually some tasks requires a subject to react as faster as possible to some events, and successively, the reaction timea are gathered.

To do so, the KbWait time is collected, saved and processed.

Chapter 10 – Sound, Keyboard and Mouse

Reaction time detection

Due to the relevance of the setting, hereafter a simplest code is provided. This code has to be fully understood.

Reaction time detection	Comments
[pippo,rect]=Screen('OpenWindow',0,[],[100 100 700 700]); *DrawFormattedText(pippo,'PRESS ANY KEY TO PROCEED','center','center');* *Screen('Flip',pippo);* *KbWait;*	
ntrials=5;	Five trials are programmed
rt=zeros(ntrials,1); *for i=1:ntrials* *WaitSecs(1);*	
Screen('FrameOval',pippo,[255 255 255], CenterRect([0 0 10 10],rect));	Draw a oval and show it
oval_time=Screen('Flip',pippo); *Screen('FillRect',pippo,[255 0 0],CenterRect([0 0 50 50],rect));* *Screen('Flip',pippo,oval_time+1+rand);*	Draw a rectangle and show it AFTER 1+rand seconds of the oval show
t0=GetSecs; *[t1, trash0]=KbWait;* *rt(i)=t1-t0;* *Screen('Flip',pippo);*	Time of the key press is gathered and subtracted to the time the rectangle is shown in order to obtain the reaction time. The screen is then cleared.

Chapter 10 – Sound, Keyboard and Mouse

Choice Reaction time **(read)**

Go/No-Go reaction time **(read)**

A simple modification to the previous listing can be done to obtain the reaction time according to a key pressed choice. As an example we can require to press R or Gas fast as possible according to the randomly shown red or green circle.

Reaction time within a video clip **(read)**

Chapter 10 – Sound, Keyboard and Mouse

Mouse Input

the mouse is a valuable tool to get inputs and information from a subject. The main functions that manage its inputs are:

- [x,y,button]=GetMouse() : (x,y) is the mouse position, while button is a boolean vector with as many elements as the number of buttons in the mouse. The elements are all 0 but that corresponding to the pressed button.
- [numclicks,x,y,button] = GetClicks : as get mouse, with a first output to save the number of clicks info.
- SetMouse(x,y) : set the mouse in position (x,y)
- HideCursor, Showcursor

Chapter 10 – Sound, Keyboard and Mouse

Mouse Input A simple example

Reaction time detection	Comments
[pippo,rect]=Screen('OpenWindow',0); *HideCursor;* *WaitSecs(3);* *ShowCursor;* *WaitSecs(3);*	*The cursor is hidden and shown later*
for i = 1:3 *SetMouse(rect(3)/2,rect(4)/4,pippo);* *WaitSecs(2);* *end*	*Three times the cursor il placed in a fix position*
[clicks,x,y,button]=GetClicks; *Screen('CloseAll');*	*A click is waited and the position is saved*

In Exercise 'Animals' add the possibility of answering by clicking with the mouse on two red and green rectangles on the left and on the right of the word. Furthermore, take care of the reaction time of each player.

Read till the end of the chapter

Chapter 10 – Sound, Keyboard and Mouse

Exercise Animals	
word=["cat","mouse","dog","bottle","pen","cup";"a","a","a","o","o","o"]; *[pippo,rect]=Screen('OpenWindow',0,[0 255 0],[100 100 500 500])* *DrawFormattedText(pippo,'PRESS ANY KEY TO PROCEED','center','center');* *Screen('Flip',pippo);* *KbWait;* *x=rect(3)/2;* *y=rect(4)/2;* *results=zeros(1,10);*	*for i= 1:10* *j=randi(6);* *MyCheers=char(word(1,j))* *Screen('DrawText',pippo,MyCheers,x,y,[255,0,0]);* *Screen('Flip',pippo);* *[tmp,code]=KbWait;* *pause(0.5);* *if code(KbName(char(word(2,j))))==1* *results(i)=1;* *end* *end* *Screen('CloseAll');* *disp(results);*

Chapter 10 – Sound, Keyboard and Mouse

Mouse click on two buttons and get the answer	Create two buttons on a [0 0 400 400] rectangle
see code below (left)	see code below (right)

```
 script bottoni
[clicks,x,y,button]=GetClicks;
    disp([x y]);
if ((x>50)&(x<100)&(y>300)&(y<350))
    answr="a";
elseif ((x>300)&(x<350)&(y>300)&(y<350))
    answr="o";
else
    answr="e";
end
    if (answr==word(2,j))&(rtime<2)
        results(1,i)=1;
        results(2,i)=rtime;
    end
```

```
 push1=[50, 300, 100, 350];
   push2=[300, 300, 350, 350];
   Screen('FillOval',pippo,[0 255 0],push1);
   Screen('DrawText',pippo,'A',60, 305);
   Screen('TextColor',pippo,[0 0 255]);
   Screen('FillOval',pippo,[255 0 0],push2);
   Screen('DrawText',pippo,'O',310, 305);
```