



# COMPUTER SKILLS

## LESSON 15

---

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

# Objectives of this lesson

We'll discuss

- Debugging techniques (chapter 6)
- String manipulation (chapter 7)

# Debugging techniques

- Any error in a computer program is called **bug**
- The process of **finding errors** in a program and **correcting** them is called **debugging**
- Error types
  - Syntax errors
  - Runtime errors
  - Logical errors

# Syntax errors

- Mistakes in using the MATLAB language, e.g.

```
>> mystr='how are you;
```

```
    mystr='how are you;
```

```
    |
```

```
Error: A MATLAB string constant is not terminated  
properly.
```

```
>> value=5
```

```
value =
```

```
      5
```

```
>> newvalue = valu + 3
```

```
Undefined function or variable 'valu'.
```

Did you mean:

```
>> newvalue = value + 3
```

see [SyntaxError.m](#)

# Syntax errors

Editor - /Users/Emiliano/Dropbox/Computer Skill/Lesson14-CS.Matlab/SyntaxError.m

```
SyntaxError.m
1 %This is a script to show that the editor highligh sintax error
2 mystr='how are you;
3
4 mystr='how are you';
5
6 value=5;
7 newvalue = valu + 3;
```

Wrong syntax

Correct syntax

Error not highlighted

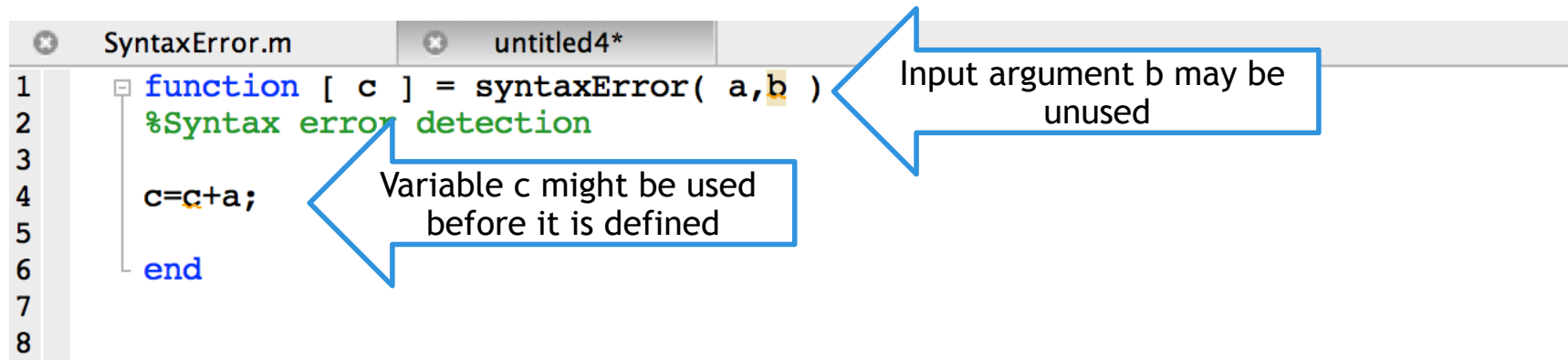
```
SyntaxError.m
1 %This is a script to show that the editor highligh sintax error
2 %mystr='how are you;
3
4 mystr='how are you';
5
6 value=5;
7 newvalue = valu + 3;
```

## Command Window

```
>> SyntaxError
Undefined function or variable 'valu'.

Error in SyntaxError (line 7)
newvalue = valu + 3;
fx >>
```

# Syntax errors



To understand the nature of the error/warning you should go with the cursor on the variable highlighted by the editor

see `SyntaxError2.m`

# Runtime/execution errors

- Errors that are found when a script or function is executing, e.g.
  - dividing by zero
  - attempting to refer an element of a vector that does not exist

Editor - /Users/Emiliano/Dropbox/Computer Skill/Lesson14-CS.Matlab/runtimeEr

```
runtimeError.m
1 %Run time error
2 vec=3:5;
3 for i=1:4
4     disp(vec(i));
5 end
6
```

Command Window

```
>> runtimeError
3
4
5
```

Attempted to access vec(4); index out of bounds because numel(vec)=3.

Error in runtimeError (line 4)  
disp(vec(i));

fx >>

```
>> a=1:3;
```

```
>> b=4:5;
```

```
>> a*b
```

Error using \*  
Inner matrix dimensions must agree.

see runtimeError.m

# Logical errors

- The more difficult to locate
- A logical error is a mistake in reasoning by the programmer
- To prove the correct functional behaviour of your code you should
  - Define a set of use cases (different combinations of input values) as much detailed as possible
  - Check the correctness of the answer/solution



# Tracing

- How to exactly know which statements have been executed
- Let us consider the following example:

```
1 function testifelse(x)
2 % testifelse will test the debugger
3 % Format: testifelse(Number)
4
5 if 3 < x < 6
6     disp('In middle of range')
7 else
8     disp('Out of range')
9 end
10 end
```

Command Window

```
>> testifelse(4)
In middle of range
>> testifelse(7)
In middle of range
>> testifelse(-2)
In middle of range
>>
```



see testifelse.m

# echo function

```
>> help echo
```

**echo** Display statements during function execution.

**echo ON** turns on echoing of commands inside Script-files.

**echo OFF** turns off echoing.

**echo file ON** where 'file' is a function name causes the named Function-file to be echoed when it is used.

**echo file OFF** turns it off.

**echo file** toggles it.

**echo ON ALL** turns on the echoing of commands inside any

Function-files that are currently in memory (i.e., the functions returned by INMEM).

**echo OFF ALL** turns them all off.

...

The screenshot shows a MATLAB Editor window with the title bar 'Editor - /Users/Emiliano/Dropbox/Computer Skill/Less...'. The editor has three tabs: 'd.m', 'printrectarea.m', and 'printem.m'. The active tab is 'd.m', which contains the following MATLAB code:

```

1 function [c]=testEcho(a,b)
2 %test of echo comand
3
4 c=a.*b;
5 if a > 0
6     c=c+1;
7 end
8 if b < 0
9     c=10000;
10 else
11     c=c+1;
12 end
13

```

Below the editor is the 'Command Window'. It shows the following commands and output:

```

>> echo testEcho on
>> testEcho(3,10)
%test of echo comand
c=a.*b;
if a > 0
    c=c+1;
end
if b < 0
else
    c=c+1;
end

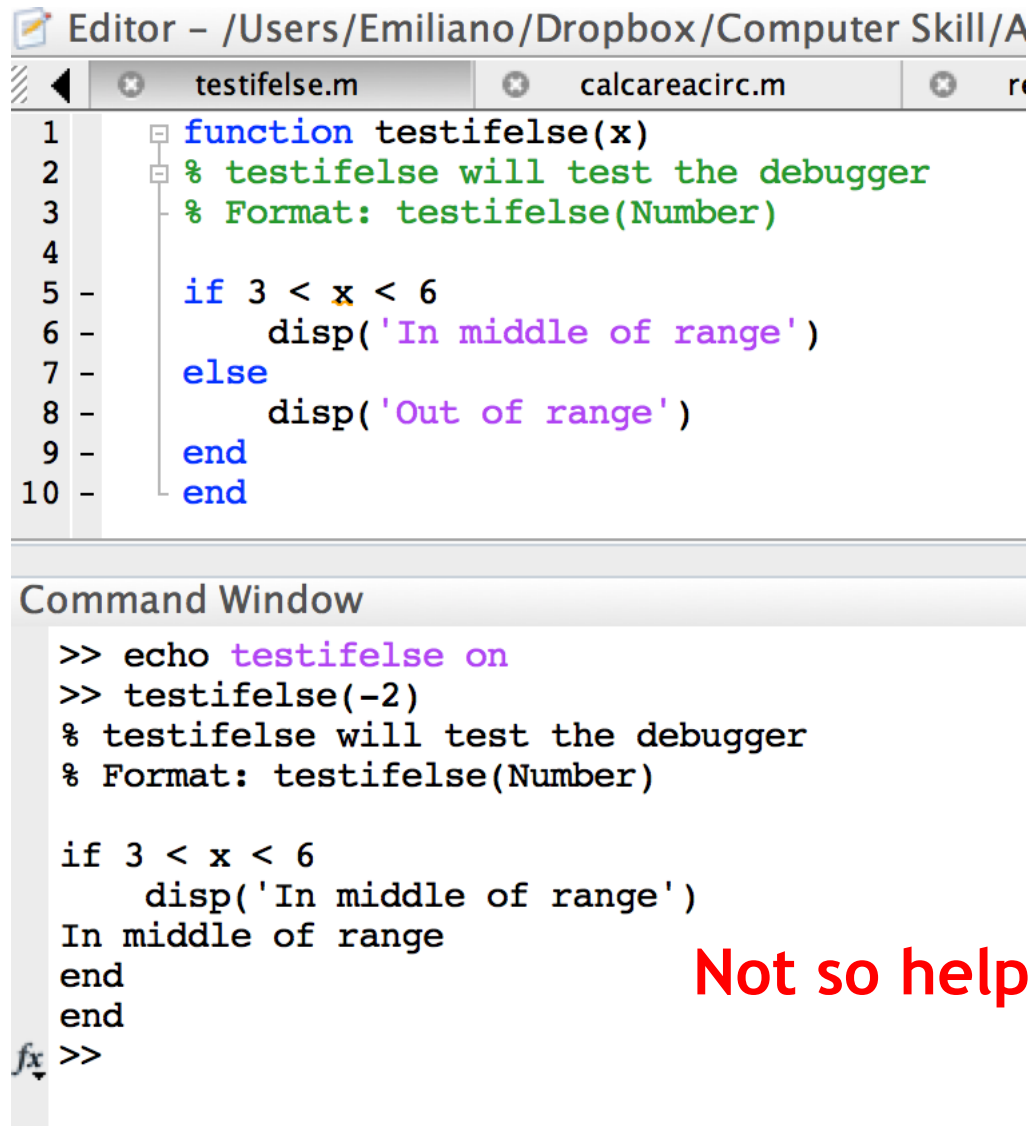
ans =

    32

```

At the bottom of the Command Window, there is a red box containing the text 'see testEcho.m'.

# Example



The image shows a MATLAB Editor window with a file named `testifelse.m`. The code defines a function `testifelse(x)` that checks if `x` is between 3 and 6. It includes comments and uses `disp` to display messages. Below the editor is the Command Window showing the execution of the function with `-2` as input. The output shows the function's internal logic and the message 'In middle of range'.

```
Editor - /Users/Emiliano/Dropbox/Computer Skill/A
testifelse.m  calcareacirc.m  r

1  function testifelse(x)
2  % testifelse will test the debugger
3  % Format: testifelse(Number)
4
5  if 3 < x < 6
6      disp('In middle of range')
7  else
8      disp('Out of range')
9  end
10 end

Command Window

>> echo testifelse on
>> testifelse(-2)
% testifelse will test the debugger
% Format: testifelse(Number)

if 3 < x < 6
    disp('In middle of range')
In middle of range
end
end
fx >>
```

Not so helpful !!!

# Code inspection

- We can set breakpoints in the code and then, from the breakpoint, we can
  - execute the code step-by-step
  - inquire the local workspace variables

$3 < x$  returns 0 because  $x = -2$

$(3 < x = 0) < 6$  is true because  $0 < 6$

Also if  $x = 4$  we have

$3 < x = 1 < 6$  is true

```
testifelse.m
1  function testifelse(x)
2  % testifelse will test the debugger
3  % Format: testifelse(Number)
4
5  if 3 < x < 6
6  disp('In middle of range')
7  else
8  disp('Out of range')
9  end
10 end
```

## Command Window

```
>> testifelse(-2)
% testifelse will test the debugger
% Format: testifelse(Number)

if 3 < x < 6
    disp('In middle of range')
6    disp('In middle of range')
K>> x

x =

    -2

K>> 3<x
ans =

     0

K>> 3<x<6
ans =

     1

fx K>>
```

# STRINGS

---

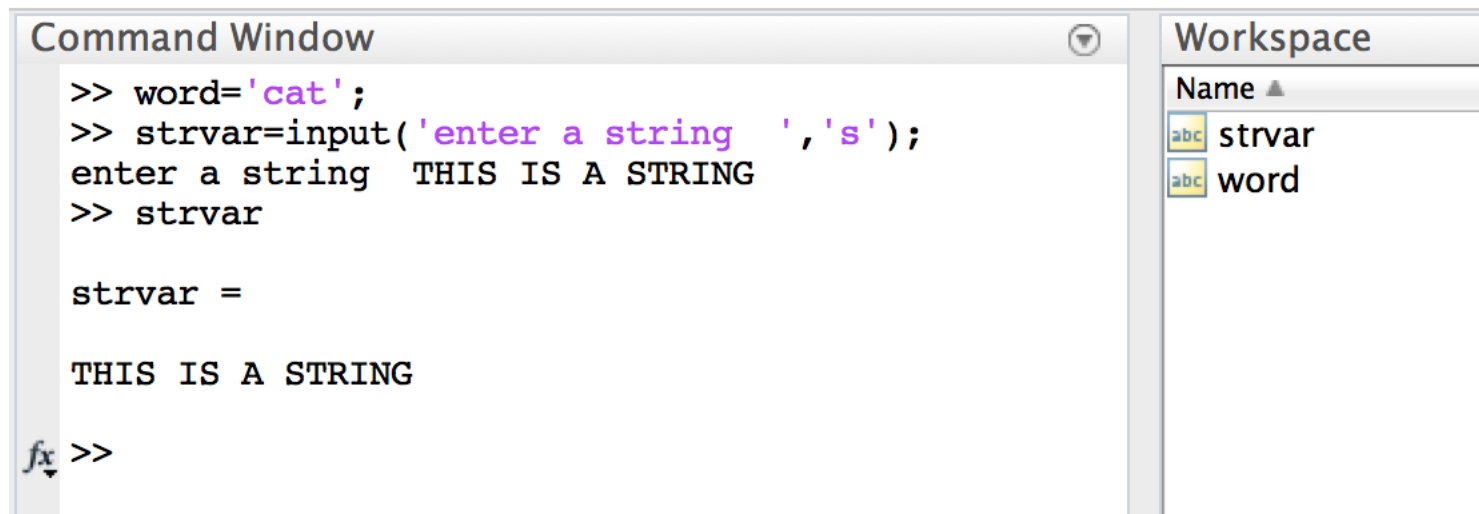
# String manipulation

- String: any number of characters contained within single quotes
- Strings are vectors in which every element is a single character
  - Vector operations mainly work also with strings
- Examples
  - **Strings:**
    - `' '`
    - `'x'`
    - `'cat'`
    - `'Hello there'`
    - `'123'`
  - **Substrings:**
    - `'there'` is a substring of `'Hello there'`

# Type of characters

- A string can contain
  - letters
  - digits
  - punctuation marks
  - white spaces
  - control characters
- Control characters are not printed but accomplish a task (e.g., tab, new line, cr)

# How to create string variables



The image shows a screenshot of the MATLAB Command Window and Workspace. The Command Window on the left contains the following code and output:

```
>> word='cat';  
>> strvar=input('enter a string ', 's');  
enter a string THIS IS A STRING  
>> strvar  
  
strvar =  
  
THIS IS A STRING  
fx >>
```

The Workspace on the right shows two variables:

Workspace	
Name ▲	
abc	strvar
abc	word



# Strings as vectors

```
>> length(word)
ans =
     3
>> length('cat')
ans =
     3
>> length(' ')
ans =
     1
>> length('')
ans =
     0
```

```
>> mystr='Hi'
mystr =
Hi
>> mystr(1)
ans =
H
>> mystr'
ans =
H
i
>> sent = 'Hello there'
sent =
Hello there
>> sent(4:8)
ans =
lo th
```

# Matrix of strings

```
>> wordmat=[ 'Hello'; 'Howdy']
wordmat =
Hello
Howdy
>> size( wordmat )
ans =
     2     5
>> wordmat( 2, 4 )
ans =
d
>> wordmat( 1, : )
ans =
Hello
>>
```

```
>> greetmat = ['Hello'; 'Goodbye']
```

Error using vertcat  
Dimensions of matrices being  
concatenated are not  
consistent.

```
>> greetmat = ['Hello ' ; 'Goodbye']
greetmat =
Hello
Goodbye
```

# Operation on strings

- Concatenation
- Creating customized strings
- Removing white space characters
- Changing cases
- Comparing strings
- Finding strings

# Concatenation

```
>> f='bird';  
>> l='house';  
>> [f l]  
ans =  
birdhouse  
>> strcat(f,l)  
ans =  
birdhouse
```

**strcat** and `[ ]` do not have the same behaviour

```
>> s1='xxx  ';  
>> length(s1)  
ans =  
6  
>> s2='  yyy';  
>> length(s2)  
ans =  
6  
>> [s1 s2]  
ans =  
xxx  yyy  
>> length(ans)  
ans =  
12  
>> strcat(s1,s2)  
ans =  
xxx  yyy  
>> length(ans)  
ans =  
9  
>> strcat(s2,s1)  
ans =  
yyyxxx  
>> length(ans)  
ans =  
9
```



remove trailing blanks

# Concatenation (cont'd)

- Char

```
>> clear greatmat
```

```
>> greatmat=char('Hello','Goodby');
```

```
>> greatmat
```

```
greatmat =
```

```
Hello
```

```
Goodby
```

```
>> size(greatmat)
```

```
ans =
```

```
2
```

```
6
```

the size of the matrix depends on the larger string concatenated

# Creating strings of blanks

- We learn that adding blanks at the end of a string is useful for row concatenation

- `blanks()`

```
>> bs4=blanks(4);
```

```
>> size(bs4)
```

```
ans =
```

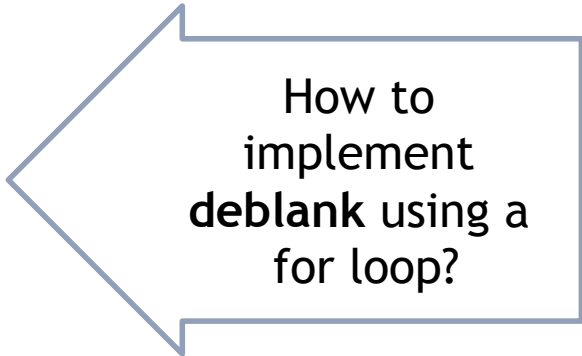
```
1      4
```

```
>> tx=['Yes We can'; 'I did ' , bs4]
```

- After adding blanks we will need also to remove them

# Removing white space characters

```
>> names=char('Sue','Cathy','Xavier');  
>> size(names)  
ans =  
      3      6  
>> n1=names(1,:)   
n1 =  
Sue  
>> length(n1)  
ans =  
      6  
>> deblank(n1)  
ans =  
Sue  
>> length(ans)  
ans =  
      3
```



How to  
implement  
**deblank** using a  
for loop?

# mydeblank

```
function [ s1 ] = mydeblank( s )
%eliminate blanks from the end of the string s and return the new
%string in s1
n=length(s);
if (size(n,1)==1 || size(n,2)==1)
    for i=1:n
        if s(i)~=' '
            s1(i)=s(i);
        else
            break;
        end
    end
else
    s1=s;
    disp('I can''t remove blanks');
end
end
```



# Assignment

- The `mydeblank` function does not work properly if a string contains blanks before the end of the string:

```
mydeblank('this string is tricky  ')
```

will return `'this'` rather than `'this string is tricky'`

- You should modify the `mydeblank` function to let it work with strings that contain blank spaces

# Creating customized strings

- `sprintf`

```
>> s2=sprintf('the value is %.2f',pi)
```

```
s2 =
```

```
the value is 3.14
```

- With `sprintf` you can create your string on the basis of variable values and save it into a variable
- Read the MATLAB help for more details

# Operation on strings

- ~~Concatenation~~
- ~~Creating customized strings~~
- ~~Removing white space characters~~
- Changing cases
- Comparing strings
- Finding strings

# Changing cases

```
>> s1='Hello'
s1 =
Hello
>> s=upper(s1)
s =
HELLO
>> s3=lower(s)
s3 =
hello
```

# Comparing strings

```
>>s1='Hello';  
>>s2='Hillo';  
>> s1==s2  
ans =  
      1      0      1      1      1
```

```
>> sum(s1==s2)==length(s1)  
ans =  
      0  
>> s2(2)='e'  
s2 =  
Hello  
>> sum(s1==s2)==length(s1)  
ans =  
      1  
>>
```

if s1 is equal to s2 the ans returning vector is filled by ones. Therefore, the sum of the vector elements will be equal to the length of the vector.

Otherwise, the two are different.

## Comparing strings (cont'd)

```
>> strcmp(s1,s2)
```

```
ans =
```

```
1
```

```
>> s1=[s1 '3456']
```

```
s1 =
```

```
Hello3456
```

```
>> strcmp(s1,s2) %is sensible to string  
length
```

```
ans =
```

```
0
```

# Assignment

- Design and implement a function that behaves as `strcmp`. From the MATLAB help we have that

`strcmp` Compare strings.

`TF = strcmp(S1,S2)` compares the strings `S1` and `S2` and returns logical 1 (true) if they are identical, and returns logical 0 (false) otherwise.

The function should manage strings of different size.

# Finding strings

```
>> s1='This is a string for test';  
>> strfind(s1,'s')
```

```
ans =
```

```
4      7      11     24
```

```
>> strfind(s1,'for')
```

```
ans =
```

```
18
```

The function `strfind` returns the positions of the input character, e.g. the positions in `s1` of the character `s`

or the starting position for the input substring



# Assignment

- Design and implement a function that behaves as `strfind`