Programming language of MATLAB

Introduction

- Like other computer programming languages, MATLAB has its own decision making structures for control of command execution.
- These decision making structures (often called *control flow* structures*)* include the following constructions :
  - for loops,
  - while loops,
  - if-else-end statements
  - Switch – case statements
- Control flow structures are often used in script M-files and function M-files.

if /else/ elseif

IF (MATLAB syntax)

```
if cond
        commands
end
```

Conditional statement: evaluates to true or false

IF /ELSE (MATLAB syntax)

```
if cond
    commands1
else
    commands2
end
```

ELSEIF (MATLAB syntax)

```
if cond1
          commands1
elseif cond2
          commands2
else
          commands3
end
```

- No need for parentheses: command blocks are between reserved words
- elseif has no space between else and if (one word)
- no semicolon (;) is needed at the end of lines containing if, else, end
- indentation of if blocks is not required, but facilitate the reading.
- the end statement is required

if /else/ elseif (Example)

IF

```
discr = b*b - 4*a*c;
if discr < 0
        disp('Warning: discriminant is negative, roots are imaginary');
end
```

IF /ELSE

```
discr = b*b - 4*a*c;
if discr < 0
   disp('Warning: discriminant is negative, roots are imaginary');
else
   disp('Roots are real, but may be repeated')
end
```

if /else/ elseif (Example)

ELSEIF

```
discr = b*b - 4*a*c;
if discr < 0
        disp(’Warning: discriminant is negative, roots are  imaginary’);
elseif discr == 0
        disp(’Discriminant is zero, roots are repeated’)
else
        disp(’Roots are real’)
end
```

Switch-case statement

```
%variable on workspace
switch variable
        case value1
            commands
        case value2
            commands
        case value3
            commands3
        otherwise %optional
            commands4
 end
```

- The switch statement executes groups of statements based on the value of a variable or expression.
- The keywords case and otherwise delineate the groups.
- Only the first matching case is executed:

  Unlike the C language switch statement, MATLAB switch does not fall through. If the first case statement is true, the other case statements do not execute. So, break statements are not required

- The otherwise block is optional and is executed if none of the case values match the value of variable.
- There must always be an end to match the switch.

Switch-case statement (example)

use switch statement to determine the number of days in a given month.

```
month =input("which month?")
switch month
          case {1, 3, 5, 7, 8, 10, 12}
               disp("31 days");
          case {4, 6, 9, 11}
                disp("30 days");
          case 2
               disp("28 or 29 days");
          otherwise
               disp("Invalid month");
 end
```

For loops

FOR (MATLAB syntax)

```
for var= initial  :  increment  :   final
            commands
end
```

1. The loop variable
   - ➢ Is defined as a vector var
   - ➢ Is a scalar within the command block

2. The command block
   - ➢ as needed commands and comments between the **for** line and the **end**

note: ndent the loops for readability, especially when they are nested.

For loops (examples)

- Example 1:

```
for ii=1:5
        x=ii*ii              % same as (1:5).^2
end
```

- Example 2:

```
n = 5;
for j=2:n
        for i=1:j-1
           A(i,j)=i/j;
           A(j,i)=i/j;
        end
end
```

```
A=
      0       0.50      0.33      0.25      0.20
   0.50          0      0.67      0.50      0.40
   0.33       0.67         0      0.75      0.60
   0.25       0.50      0.75         0      0.80
   0.20       0.40      0.60      0.80         0

                    Isequal( A,A’)
                       ans=  1
```

While loops

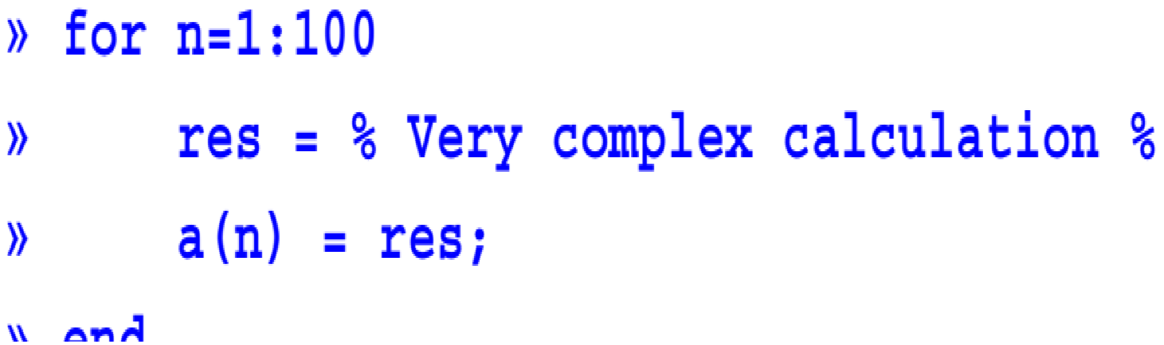
while (MATLAB syntax)

```
while cond
commands
end
```

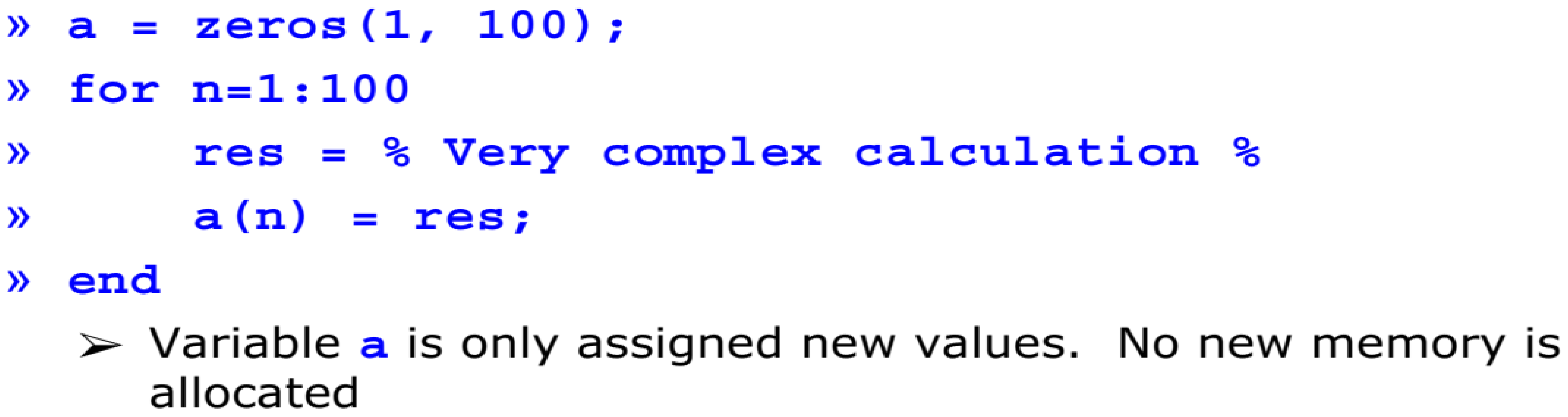
- The command block will execute while the conditional expression is true
  - You can use **break** to exit a loop

Preallocation

Consider this block which creates a vector a , of 100 elements, element by element.



```
» for n=1:100
»      res = % Very complex calculation %
»      a(n) = res;
» end
```



```
» a = zeros(1, 100);
» for n=1:100
»       res = % Very complex calculation %
»       a(n) = res;
» end
```

- ➢ Variable a is only assigned new values. No new memory is allocated

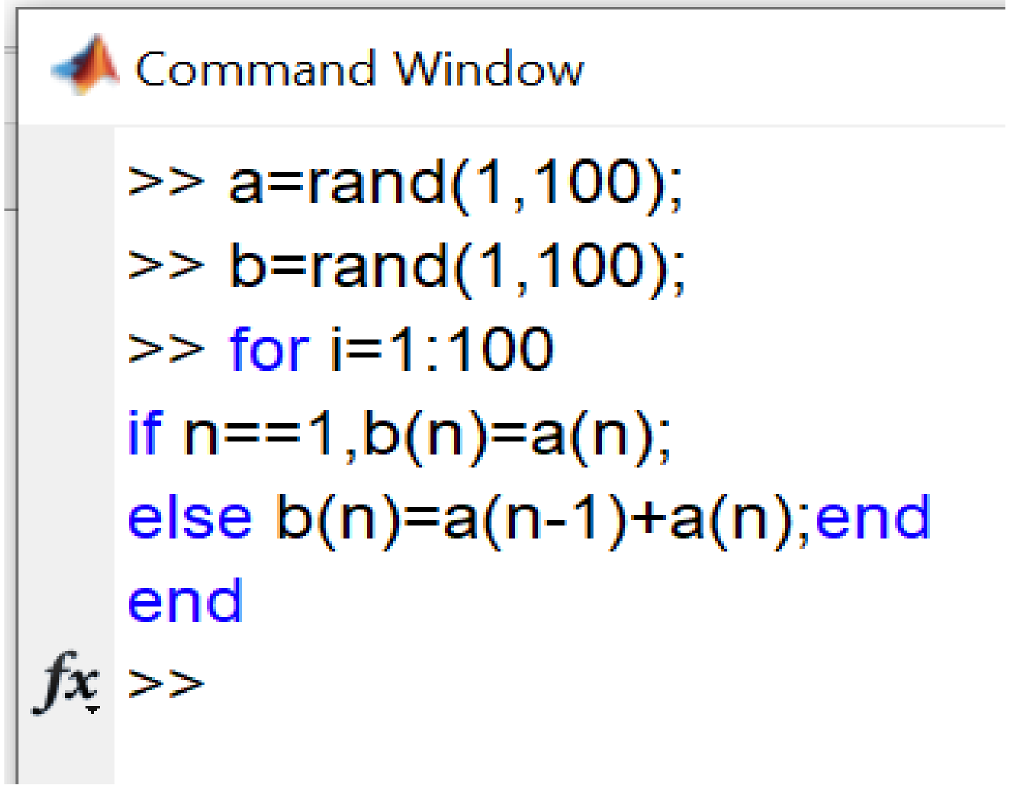
Vectorization

- Vectorized code is more efficient for MATLAB (LAB session N°1)
- Use indexing and matrix operations to avoid loops
- example: to add every two consecutive termsof a vector:

```
» a=rand(1,100);
» b=zeros(1,100);
» for n=1:100
»     if n==1
»         b(n)=a(n);
»     else
»         b(n)=a(n-1)+a(n);
»     end
» end
```

```
» a=rand(1,100);
» b=[0 a(1:end-1)]+a;
```

Programs in interactive mode



```
>> a=rand(1,100);
>> b=rand(1,100);
>> for i=1:100
if n==1,b(n)=a(n);
else b(n)=a(n-1)+a(n);end
end
>>
```

Programs using MATLAB Editor

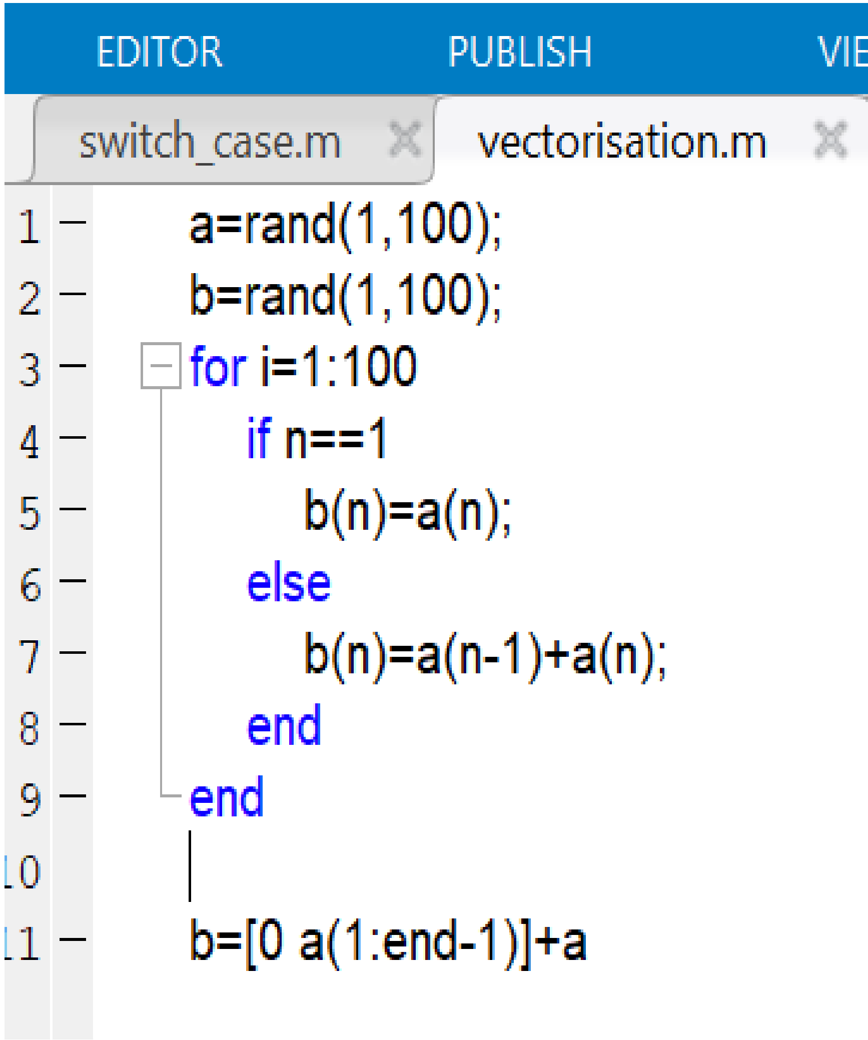
Use the MATLAB *editor* to create a file;

File → New → M-file.

Enter the statements ,

Save the file, for example, vectorization.m

M-file script →



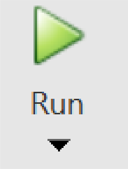
```
a=rand(1,100);
b=rand(1,100);
for i=1:100
    if n==1
        b(n)=a(n);
    else
        b(n)=a(n-1)+a(n);
    end
end

b=[0 a(1:end-1)]+a
```

M-Files Scripts

- A *script file* is file that contains a sequence of MATLAB statements.
- Script files have a filename extension .m and are often called m-files.
- Run script files using:
  - Filename on Command Window

    ```
    >>vectorisation
    ```

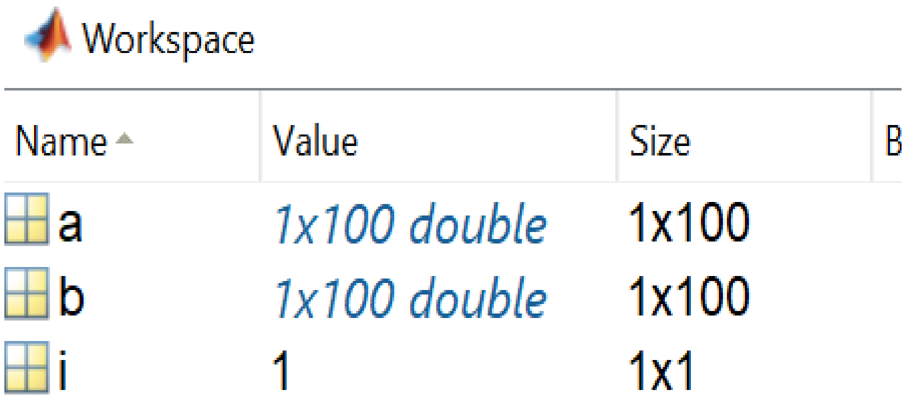
  - Run icon on the editor tool bar



- After m-file runing,

(a, b, and i) variables appear

on the workspace.



Name	Value	Size	B
a	*1x100 double*	1x100	
b	*1x100 double*	1x100	
i	1	1x1	

M-Files Scripts

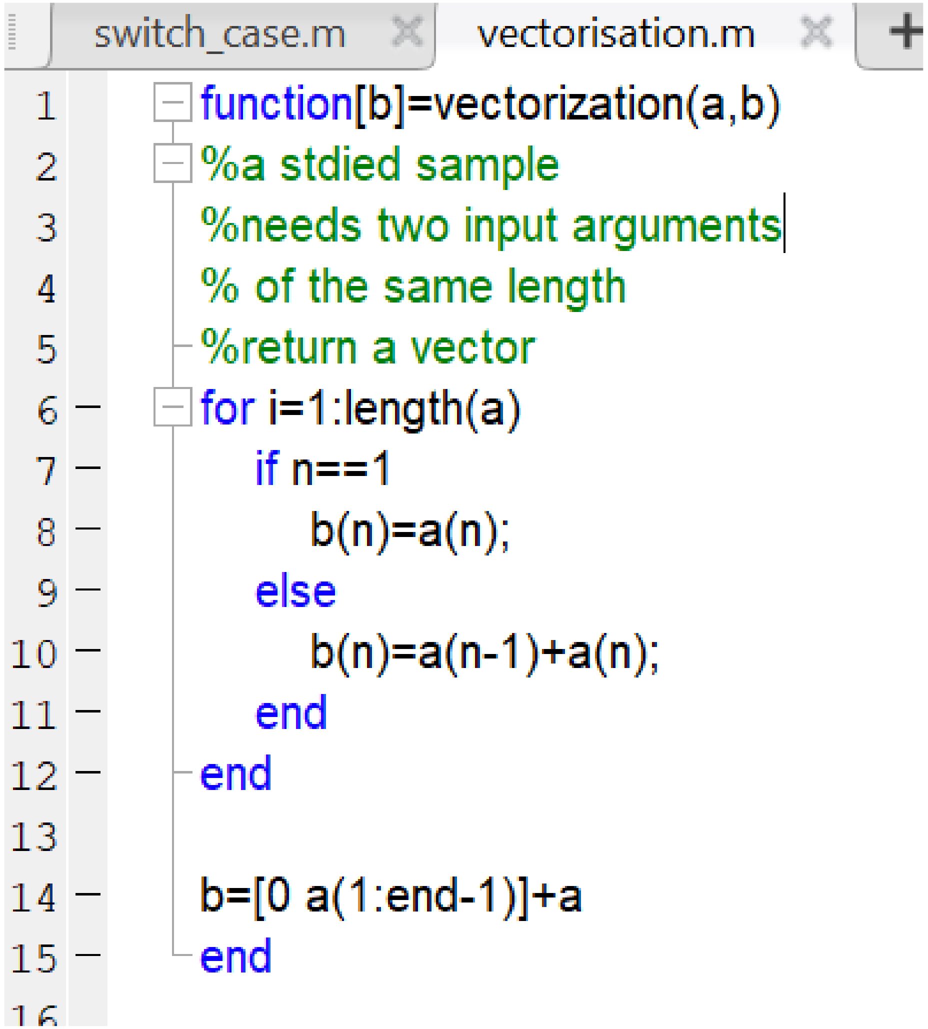
All variables created in a script file are added to the workspace. This may have undesirable effects, because:

- Variables already existing in the workspace may be overwritten.
- The execution of the script can be affected by the state variables in the workspace.

As a result, because scripts have some undesirable side-effects, it is better to code applications (specially complicated ones) using rather function M-file.

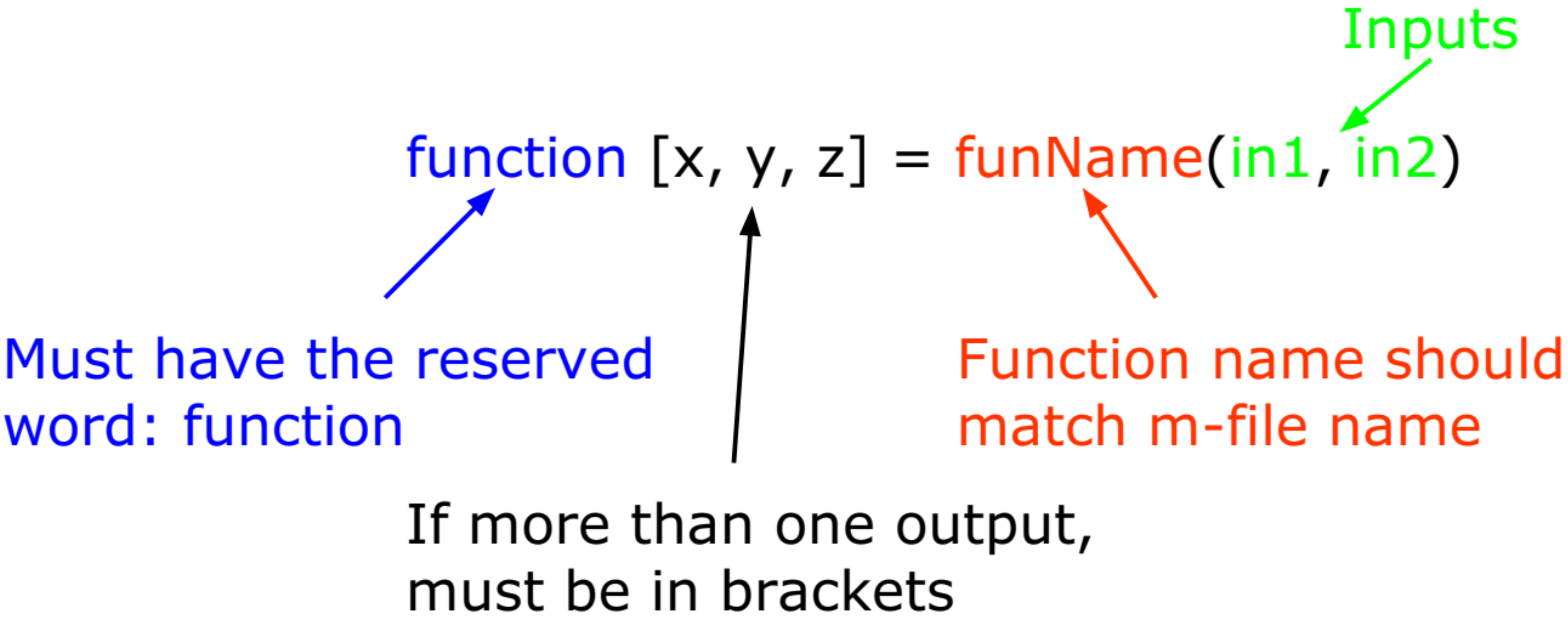
User-defined functions

- Functions look exactly like scripts, but for one difference
  ➢ Functions must have a function declaration



```
function[b]=vectorization(a,b)
%a stdied sample
%needs two input arguments
% of the same length
%return a vector
for i=1:length(a)
    if n==1
        b(n)=a(n);
    else
        b(n)=a(n-1)+a(n);
    end
end

b=[0 a(1:end-1)]+a
end
```

User-defined functions



Inputs

```
function [x, y, z] = funName(in1, in2)
```

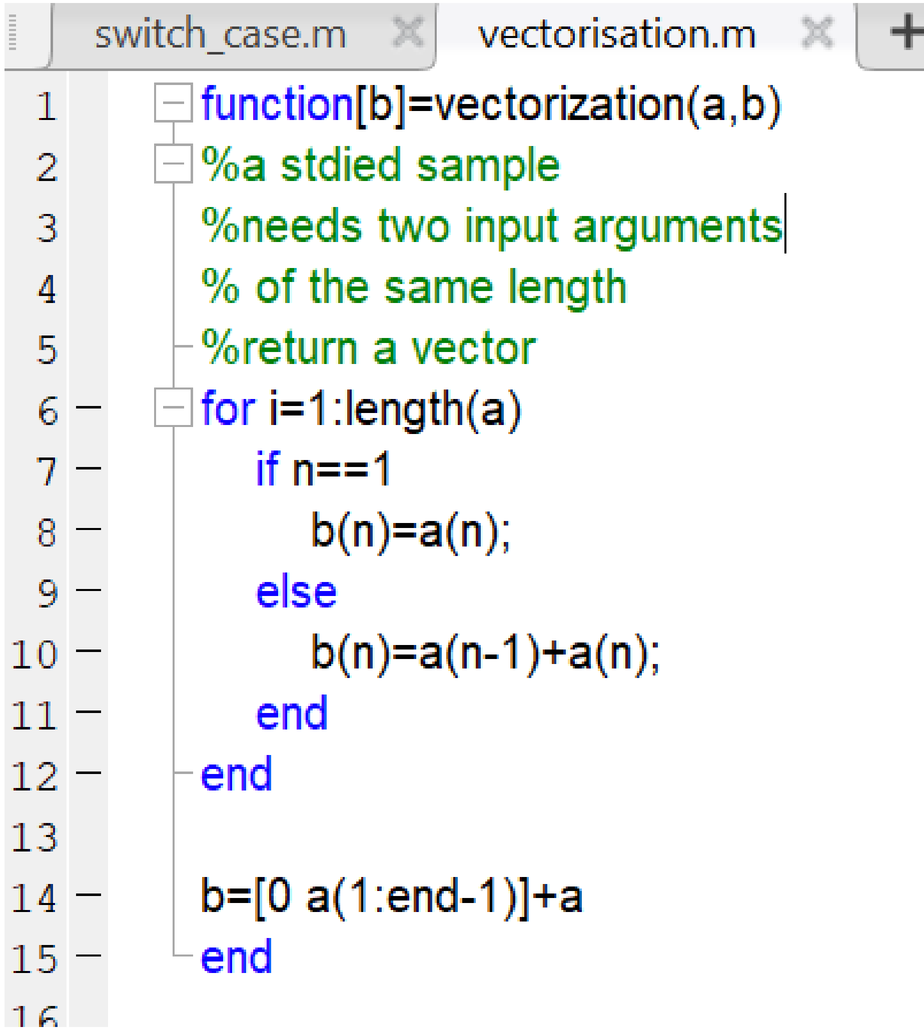
Must have the reserved word: function

Function name should match m-file name

If more than one output, must be in brackets

User-defined functions

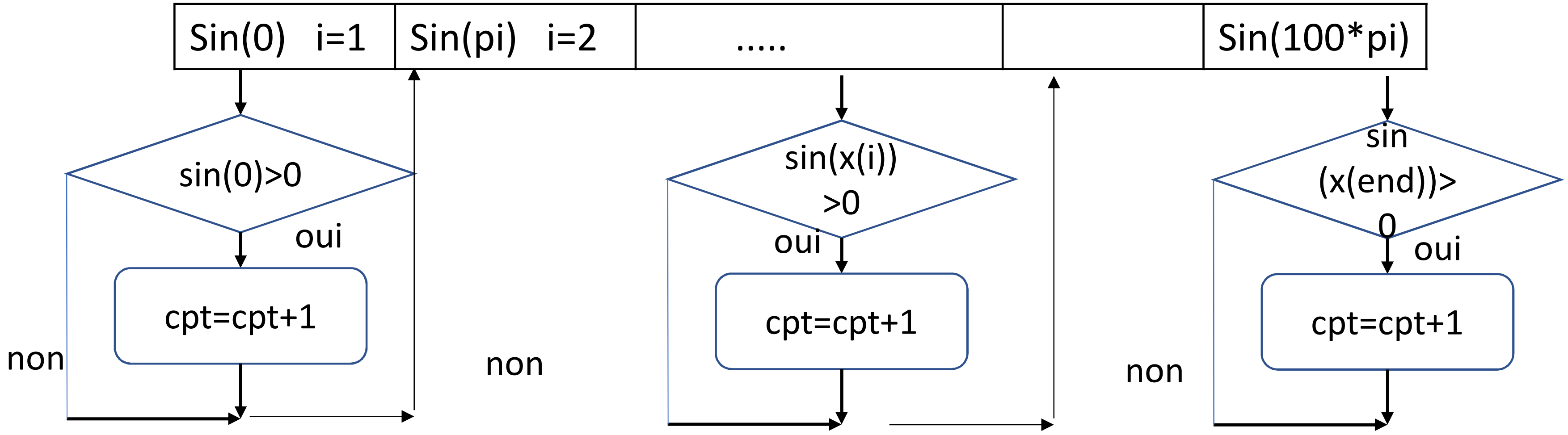
- No need for return
- Variable scope: “local variables”
- Invoking functios using their filename (script or command window)



```
function[b]=vectorization(a,b)
%a stdied sample
%needs two input arguments
% of the same length
%return a vector
for i=1:length(a)
    if n==1
        b(n)=a(n);
    else
        b(n)=a(n-1)+a(n);
    end
end

b=[0 a(1:end-1)]+a
end
```

Example

- Given X = sin(linspace(0,10*pi,100))
- How many positive entries are their in X.
- X=



Sin(0) i=1

Sin(pi) i=2

.....

Sin(100*pi)

sin(0)>0

oui

cpt=cpt+1

non

non

sin(x(i)) >0

oui

cpt=cpt+1

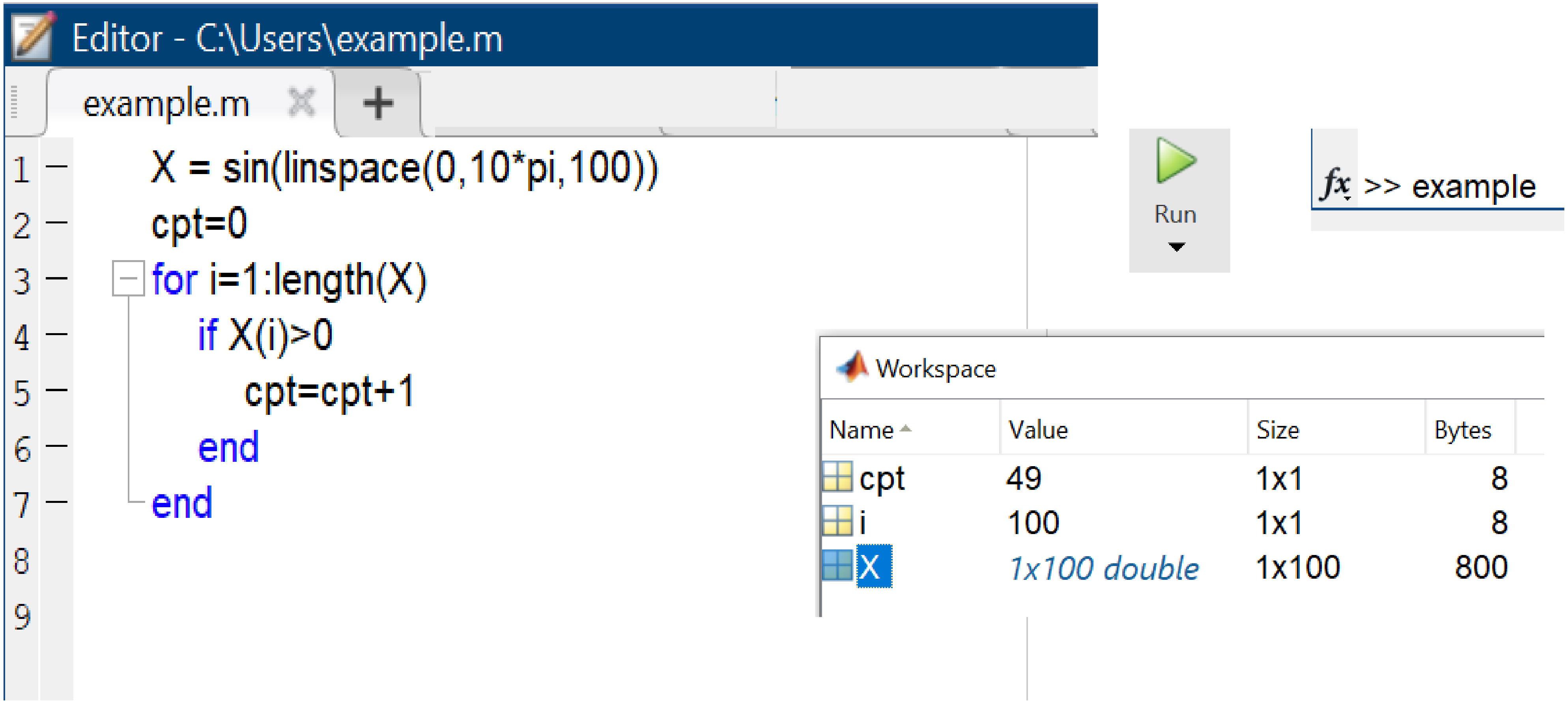
non

sin (x(end))> 0

oui

cpt=cpt+1

Example

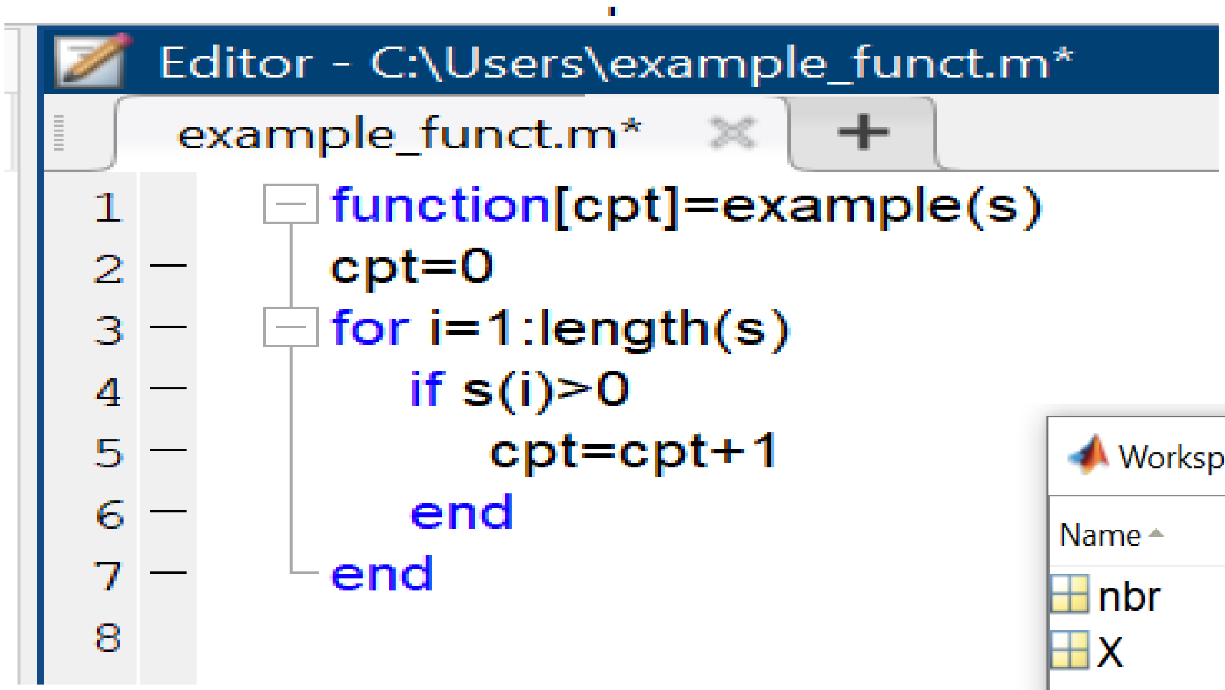


```
X = sin(linspace(0,10*pi,100))
cpt=0
for i=1:length(X)
    if X(i)>0
        cpt=cpt+1
    end
end
```

```
>> example
```

Name	Value	Size	Bytes
cpt	49	1x1	8
i	100	1x1	8
X	*1x100 double*	1x100	800

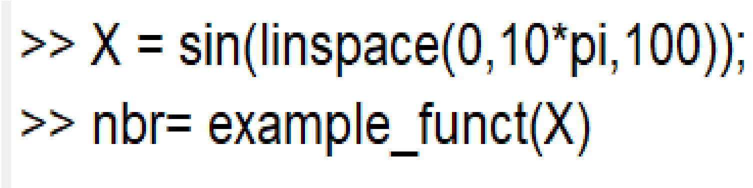
Example



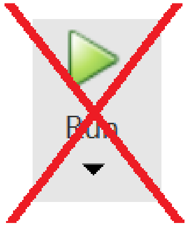
Editor - C:\Users\example_funct.m*

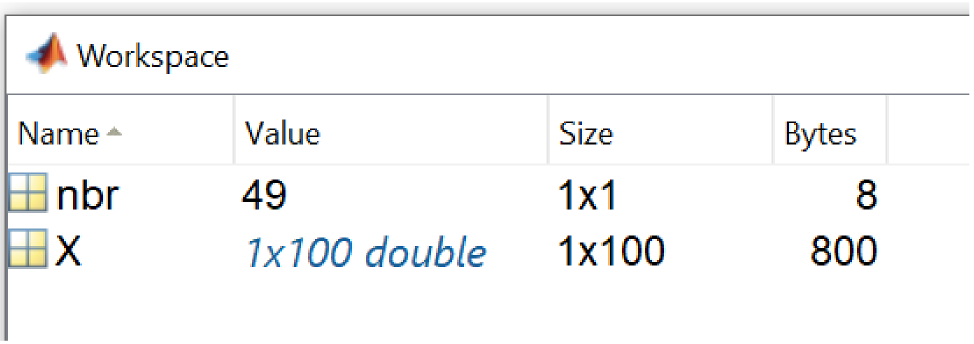
example_funct.m*

```
function[cpt]=example(s)
cpt=0
for i=1:length(s)
    if s(i)>0
        cpt=cpt+1
    end
end
```



```
>> X = sin(linspace(0,10*pi,100));
>> nbr= example_funct(X)
```





Workspace

Name	Value	Size	Bytes
nbr	49	1x1	8
X	*1x100 double*	1x100	800

Overloading

- MATLAB functions are generally overloaded :
  - ➢ Can take a variable number of inputs
  - ➢ Can return a variable number of outputs

```
>> A=randi([1 10],3,4);
>> a=randi(5,2),
>>aa=randi([-1 3])
>> B=size(A);  %vector
>> [n,m]=size(A); %2 scalars
>> size(A,1);
```

Overloading

- Users can overload their own functions by having variable number of input and output arguments

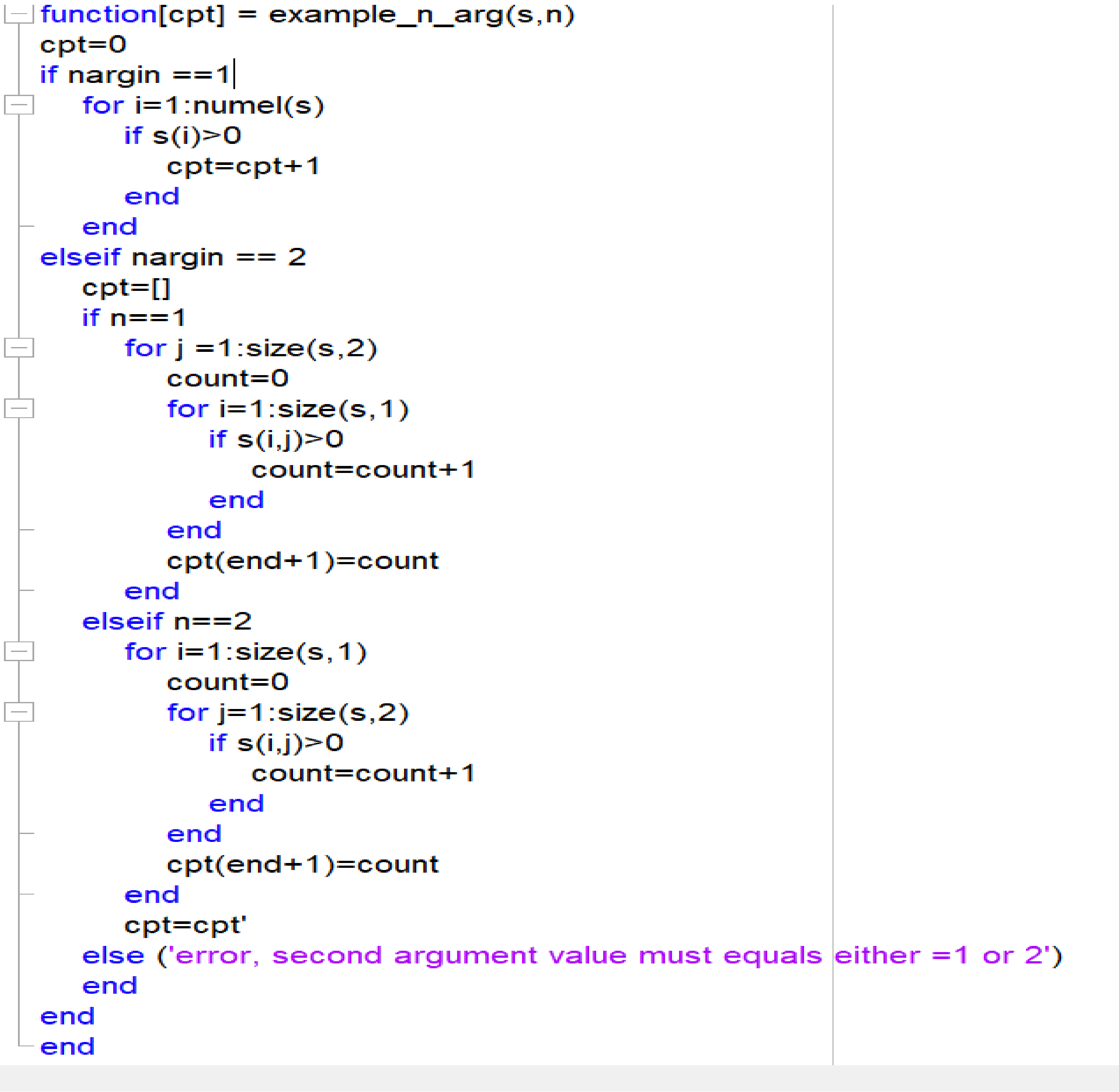
(using : Nargin, nargout, varargin, varargout, inputname , ….)

Example :

```
function myplot(x,varargin)
function [s,varargout] = mysize(x)
```

Overloading

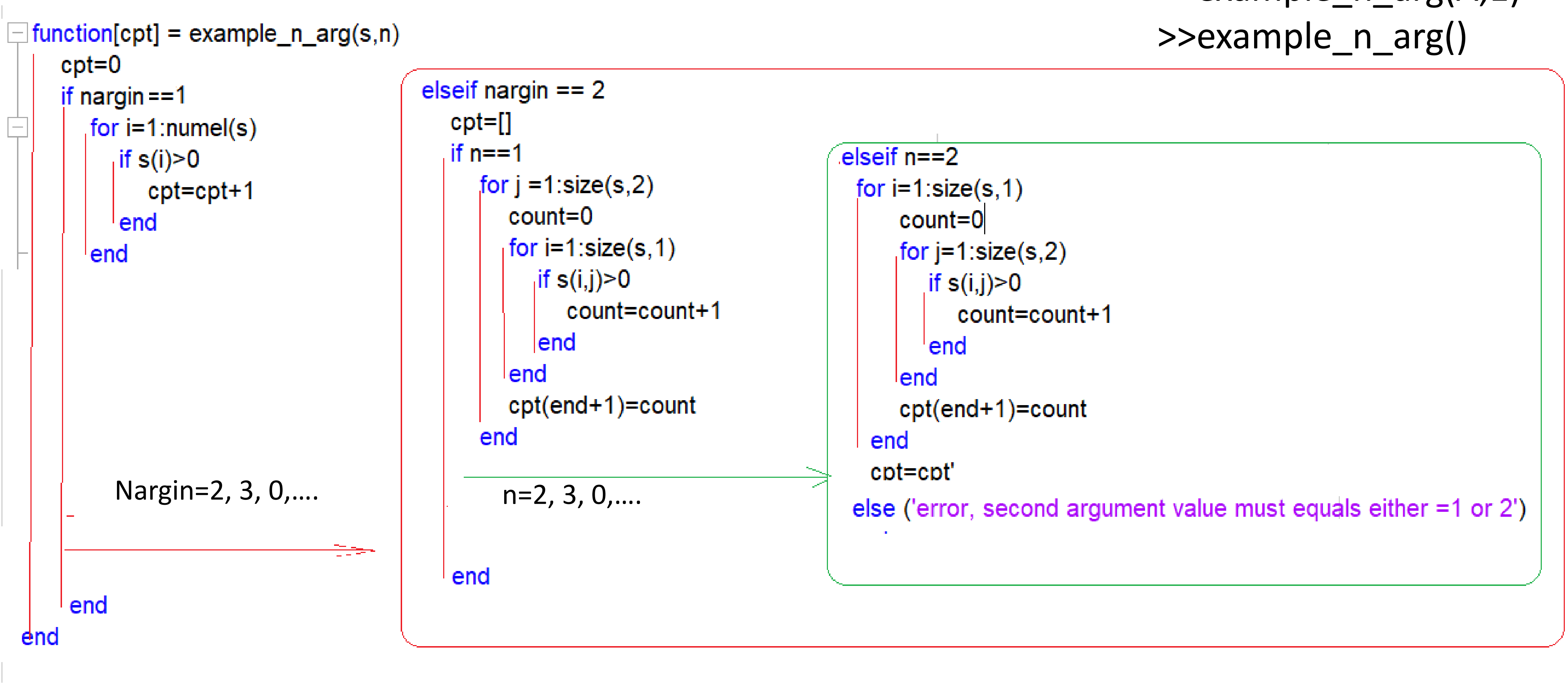
```
>>example_n_arg(A,2)
>>example_n_arg(A,1)
>>example_n_arg(A)
>>example_n_arg()
```



```
function[cpt] = example_n_arg(s,n)
cpt=0
if nargin ==1
    for i=1:numel(s)
        if s(i)>0
            cpt=cpt+1
        end
    end
elseif nargin == 2
    cpt=[]
    if n==1
        for j =1:size(s,2)
            count=0
            for i=1:size(s,1)
                if s(i,j)>0
                    count=count+1
                end
            end
            cpt(end+1)=count
        end
    elseif n==2
        for i=1:size(s,1)
            count=0
            for j=1:size(s,2)
                if s(i,j)>0
                    count=count+1
                end
            end
            cpt(end+1)=count
        end
        cpt=cpt'
    else ('error, second argument value must equals either =1 or 2')
    end
end
end
```

Overloading

```
>>example_n_arg(A)
>>example_n_arg(A,2)
>>example_n_arg(A,1)
>>example_n_arg()
```



```
function[cpt] = example_n_arg(s,n)
    cpt=0
    if nargin ==1
        for i=1:numel(s)
            if s(i)>0
                cpt=cpt+1
            end
        end
```

Nargin=2, 3, 0,....

```
    end
end
```

```
elseif nargin == 2
    cpt=[]
    if n==1
        for j =1:size(s,2)
            count=0
            for i=1:size(s,1)
                if s(i,j)>0
                    count=count+1
                end
            end
            cpt(end+1)=count
        end
```

n=2, 3, 0,....

```
    end
```

```
elseif n==2
    for i=1:size(s,1)
        count=0
        for j=1:size(s,2)
            if s(i,j)>0
                count=count+1
            end
        end
        cpt(end+1)=count
    end
    cpt=cpt'
 else ('error, second argument value must equals either =1 or 2')
```

Debbuging

Debugging is the process of identifying and fixing errors, or "bugs," in computer programs. A debugger is a software tool that helps programmers debug their code by allowing them to inspect variables, control program execution, and analyze the flow of their program. Debuggers are essential for efficiently locating and resolving issues in code.

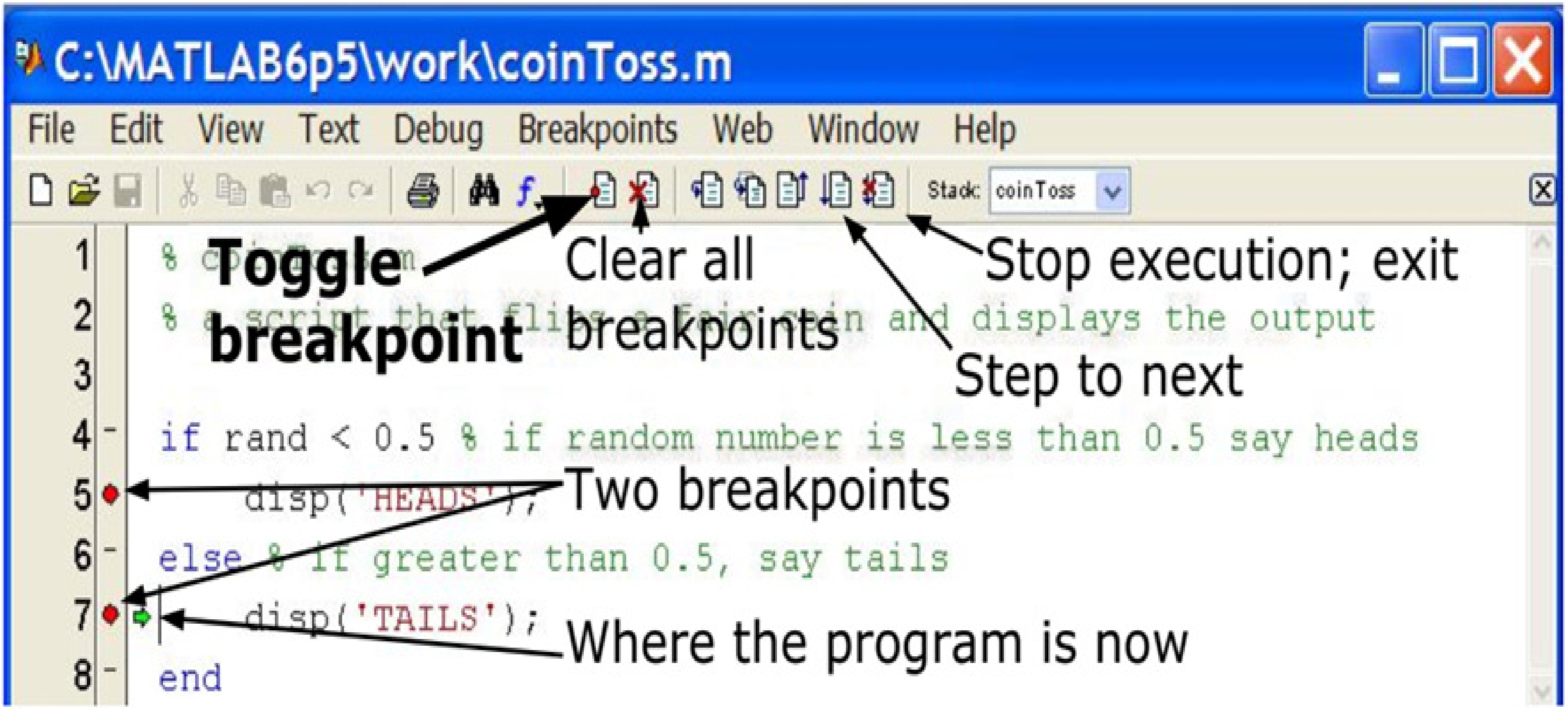
Debbuging

The MATLAB editor is both a text editor specialized for creating M-files and a graphical MATLAB debugger

To use the debugger, set breakpoints

➢ Click on – next to line numbers in m-files
➢ Each red dot that appears is a breakpoint
➢ Run the program
➢ The program pauses when it reaches a breakpoint
➢ Use the command window to probe variables
➢ Use the debugging buttons to control debugger

Debbuging


C:\MATLAB6p5\work\coinToss.m
File Edit View Text Debug Breakpoints Web Window Help
Stack: coinToss

```
% coinToss.m
% a script that flips a fair coin and displays the output

if rand < 0.5 % if random number is less than 0.5 say heads
    disp('HEADS');
else % if greater than 0.5, say tails
    disp('TAILS');
end
```

Toggle breakpoint

Clear all breakpoints

Stop execution; exit

Step to next

Two breakpoints

Where the program is now