



# COMPUTER SKILLS

## LESSON 9

---

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

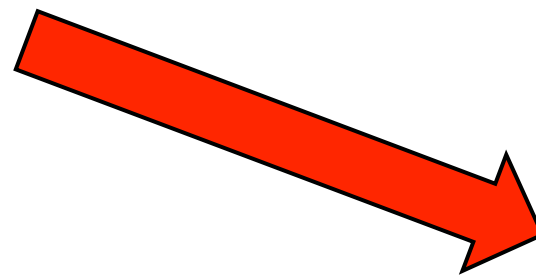
# Objectives of this lesson

We'll discuss

- Functions
- Looping statements (or **loops**): allow **to repeat** other statement(s) either a specified number of times (***counted loops***) or until a given condition is satisfied (***conditional loops***)
  - **for** loops
  - Nested **for** loops
  - **while** loops

# Programming paradigms

- **Procedural programming:**  
the basic programs or  
subprograms are sequences  
of operations on data items

 MATLAB scripts MATLAB functions

# User-defined functions

- We have already used many built-in functions in MATLAB
  - E.g., `linspace`, `size`, `length`, `disp`, `plot`
- The programmer can also define and then use functions by her/his own
- These user-defined functions can be *called* in the Command Window or in a script
  - To call a function, we use the **name of the function**, followed by the **input argument(s)** in parentheses
    - E.g., `length(vec)`
  - The result of the function is then *returned* as **output argument(s)**
    - E.g., `lv = length(vec)`
- For now, we consider user-defined functions that return a single value

# Function definition

- For now, we store every function in a separate M-file, as we already do for scripts
- A function consists of:
  - The **function header** (the first line) comprised of:
    - The reserved word **function**
    - The name of the **output argument** followed by the assignment operator (=) as the function returns a result
    - The **name of the function** (it must be the same as the name of the M-file that stores the function)
    - The input arguments in parentheses
  - A **comment** that describes what the function does
  - The **function body**, which includes all statements and eventually must put a value in the output argument
  - The reserved word **end** at the end of the function

# Example of function

- We want to define a function that creates a vector of increasing integer values from mymin to mymax
- Our function receives two input arguments (mymin and mymax) and returns a vector with values from mymin to mymax
- First, we want to be sure that mymin is less than mymax
- If not, **we need to exchange their values** before creating the vector

## Example of function (2)

Reserved word **function**

Output argument

Two input arguments

```
function outvec = createvec(mymin,mymax)
% createvec creates a vector that iterates from a
minimum to a maximum
% Returns a vector
```

Function name

Assignment

```
% if the minimum is not smaller than the maximum,
exchange the values
```

```
if mymin > mymax
    temp = mymin;
    mymin = mymax;
    mymax = temp;
```

```
end
```

```
outvec = mymin:mymax;
```

```
end
```

See createvec.m

- We save the function in the M-file createvec.m (the file name must be the same of the function)

- We call the function

```
>> createvec(7,3)
```

```
ans =
```

```
3 4 5 6 7
```

## Example of function (3)

- Note that **to exchange values**, a third variable (called a **temporary variable**) is required, otherwise we lost one of the two values!
- Assume that **mymin = 5** and **mymax = 3**
- Be careful: the following code is wrong and describes a common error!

```
mymin = mymax;           % mymin = 3
mymax = mymin;           % mymax = 3
```

- The following code is correct

```
temp = mymin;             % temp = 5
mymin = mymax;             % mymin = 3
mymax = temp;             % mymax = 5
```



# Iteration in general

- **Iteration** allows controlled repetition of a code block (also called the action of the loop)
- Control statements at the beginning of the code block specify the manner and extent of the repetition:
  - The **for loop** is designed to repeat its code block a **fixed number of times** and largely *automates* the process of managing the iteration
  - The **while loop** is more flexible: its code block can be repeated a **variable number of times** as long as a condition remains true; it is much more of a “*do-it-yourself*” iteration kit

# for loop

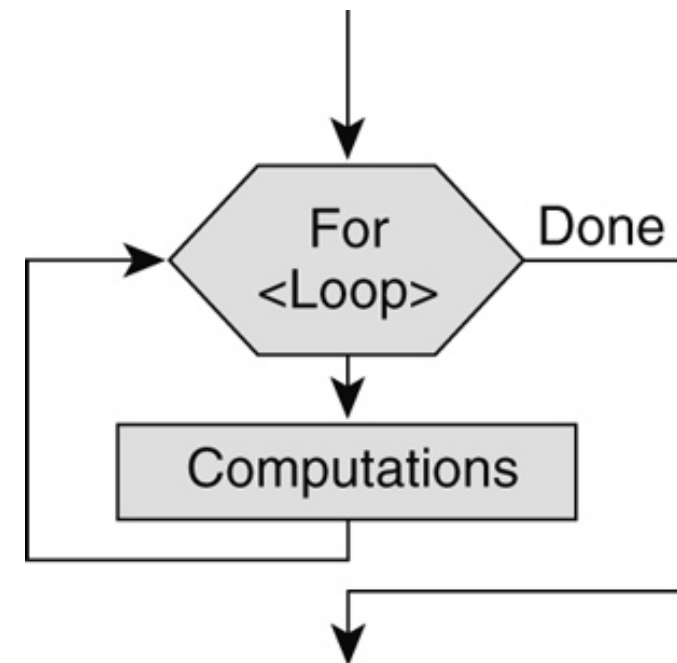
The template for the for loop is:

```
for loopvar = vector  
    <code block>  
end
```

More in general:

```
for variable = expr  
    <code block>  
end
```

The for loop automatically sets the value of the loop variable `loopvar` to each element of the vector in turn and executes the code block with that value



# Some example of for loop

```
% implement zeros(1,N)
N=10;
for i=1:N           % we specify the range of values
    v(i)=0;         % using the column operator
end
```

```
% set the odd elements of v to 1
for i=1:2:N
    v(i)=1;
end
```

```
% alternative way to code the above problem
for i=[1,3,5,7,9]
    v(i)=1;
end
```

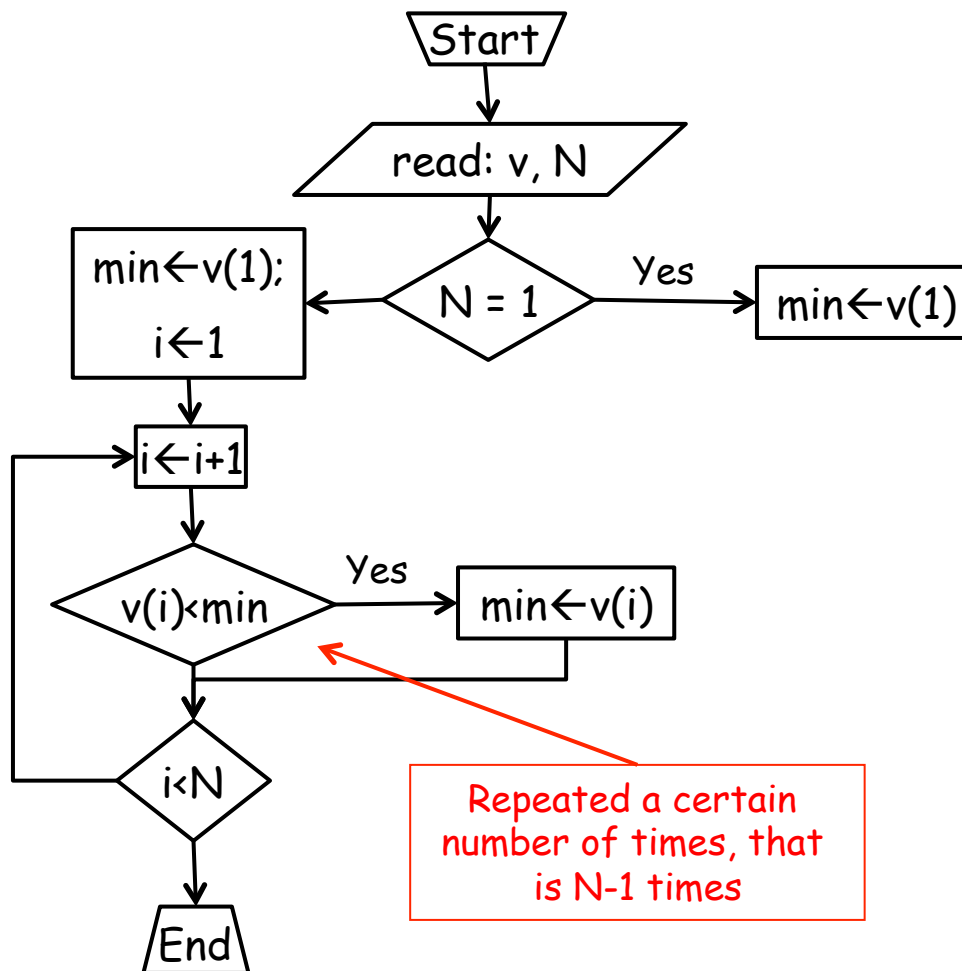
# Example: input in a for loop

- It is not always necessary to use the value of the loop variable in the code block

See forecho.m

```
% This script loops to repeat the action of  
% prompting the user for a number and echo-printing it  
  
for iv = 1:3  
    inputnum = input('Enter a number: ');  
    fprintf('You entered %.1f\n', inputnum)  
end
```

# min(v), see Lesson 5



```
v = [4 5 2 19 5 7 8];  
N = length(v);  
mymin = v(1);
```

```
for i=2:N  
    if v(i) < mymin  
        mymin = v(i);  
    end  
end
```

# Find the maximum value in a vector

%Example of for loop

```
A = randi([0,100],1,10) % initial vector
theMax = A(1);
theIndex = 1;
for index = 2:length(A)
    if A(index) > theMax
        theMax = A(index);
        theIndex = index;
    end
end
fprintf('the max value in A is %d at %d\n',
    theMax, theIndex);
```

# Find the maximum value in a vector

%Example of for loop

```
A = randi([0,100],1,10) % initial vector
theMax = A(1);          % set initial max value
for x = A                % iterate through A
    if x > theMax         % test each element
        theMax = x;
    end
end
fprintf('the max value in A is %d\n', theMax);
```

See max\_vect.m

# Nested for loops

- The action of a loop can be any valide statement(s)
- When the action of a loop is another loop, this is called a **nested loop**
- The general form of a nested loop is

```
for loopvar1 = range1
    %action1 includes the inner loop
    for loopvar2 = range2
        action2
    end
end
```



# Example of nested for loop

```
% Prints a box of stars
% How many will be specified by two variables
% for the number of rows and columns
nrows = input('Enter the number of rows: ');
ncols = input('Enter the number of columns: ');
% loop over the rows
for i=1:nrows
    % for every row loop, print '*'s and then one \n
    for j=1:ncols
        fprintf('*');
    end
    fprintf('\n');
end
```

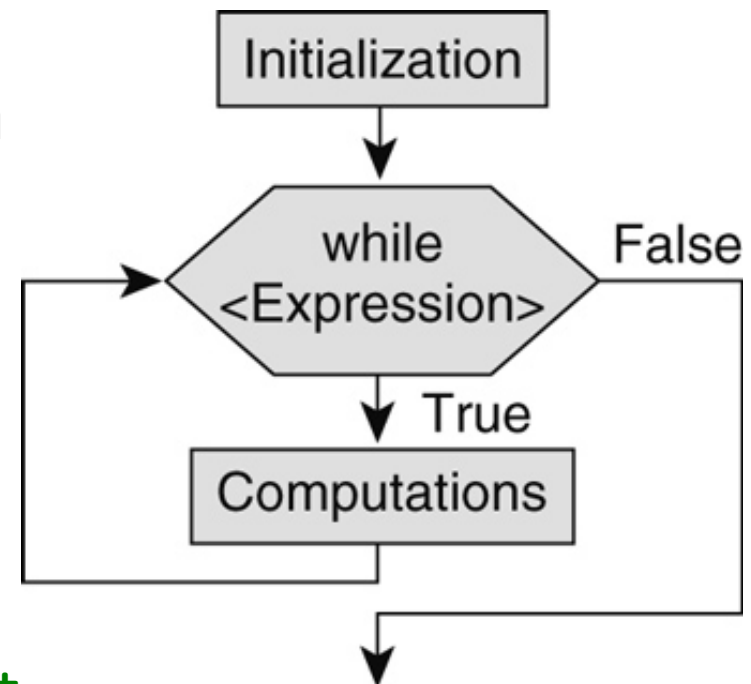
See printstars.m and printstars2.m

# while loops

The code block is repeated as long as the logical expression returns true

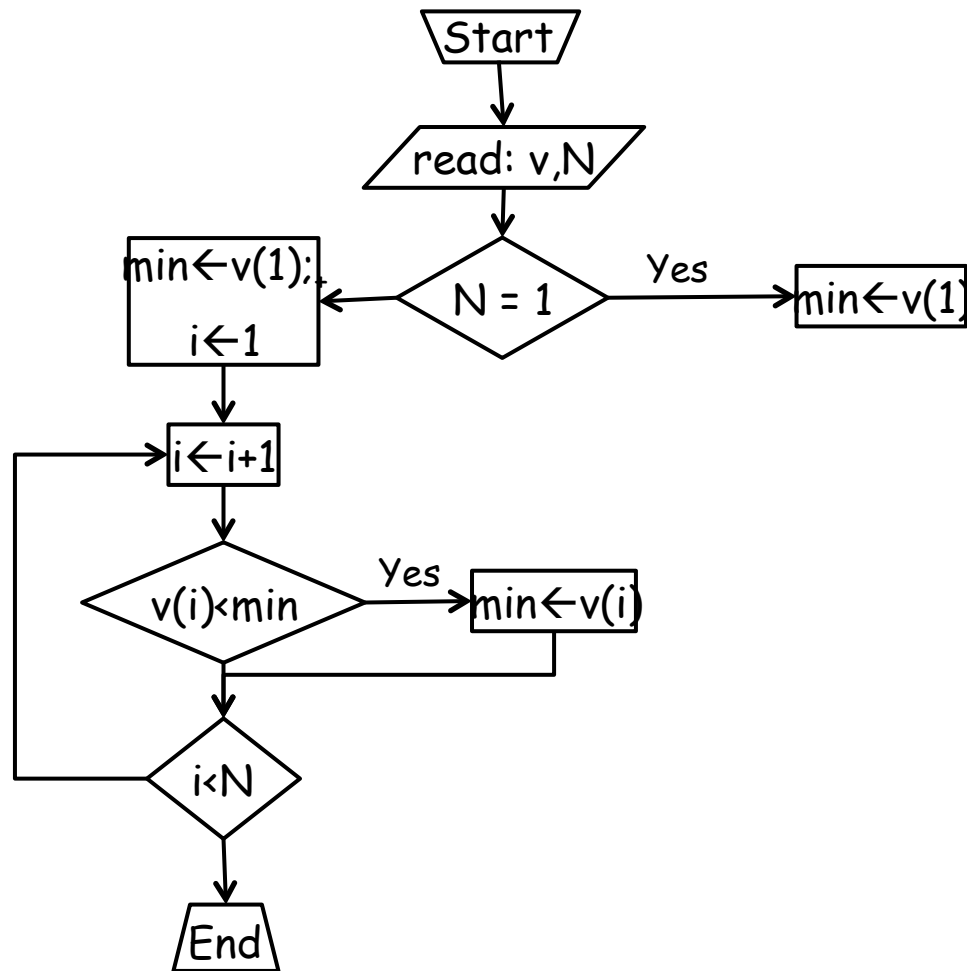
The while loop template is:

```
<initialization>  
while <logical expression>  
    <code block>  
    % must make some changes  
    % to enable the loop to terminate  
end
```



while loops can be also nested, i.e., one while loop inside another while loop

# min(v)



```

v=[4 5 2 19 5 7 8];
N=length(v);
if N==1
    mymin=v(1);
else
    mymin=v(1);
    i=2;
    while (i<=N)
        if v(i)<mymin
            mymin=v(i);
        end
        i=i+1;
    end
end
end

```

# break and continue

- You can programmatically exit a loop using a `break` statement, or skip to the next iteration of a loop using a `continue` statement
- `break` terminates the execution of `for` or `while` loop
  - Statements in the loop after the `break` statement do not execute
  - In nested loops, `break` exits only from the loop in which it occurs

Example: sum a sequence of random numbers until the next random number is greater than an upper limit

```
limit = 0.8;
s = 0;

while 1
    tmp = rand;
    if tmp > limit
        break
    end
    s = s + tmp;
end
```

# Example: quadratic equation

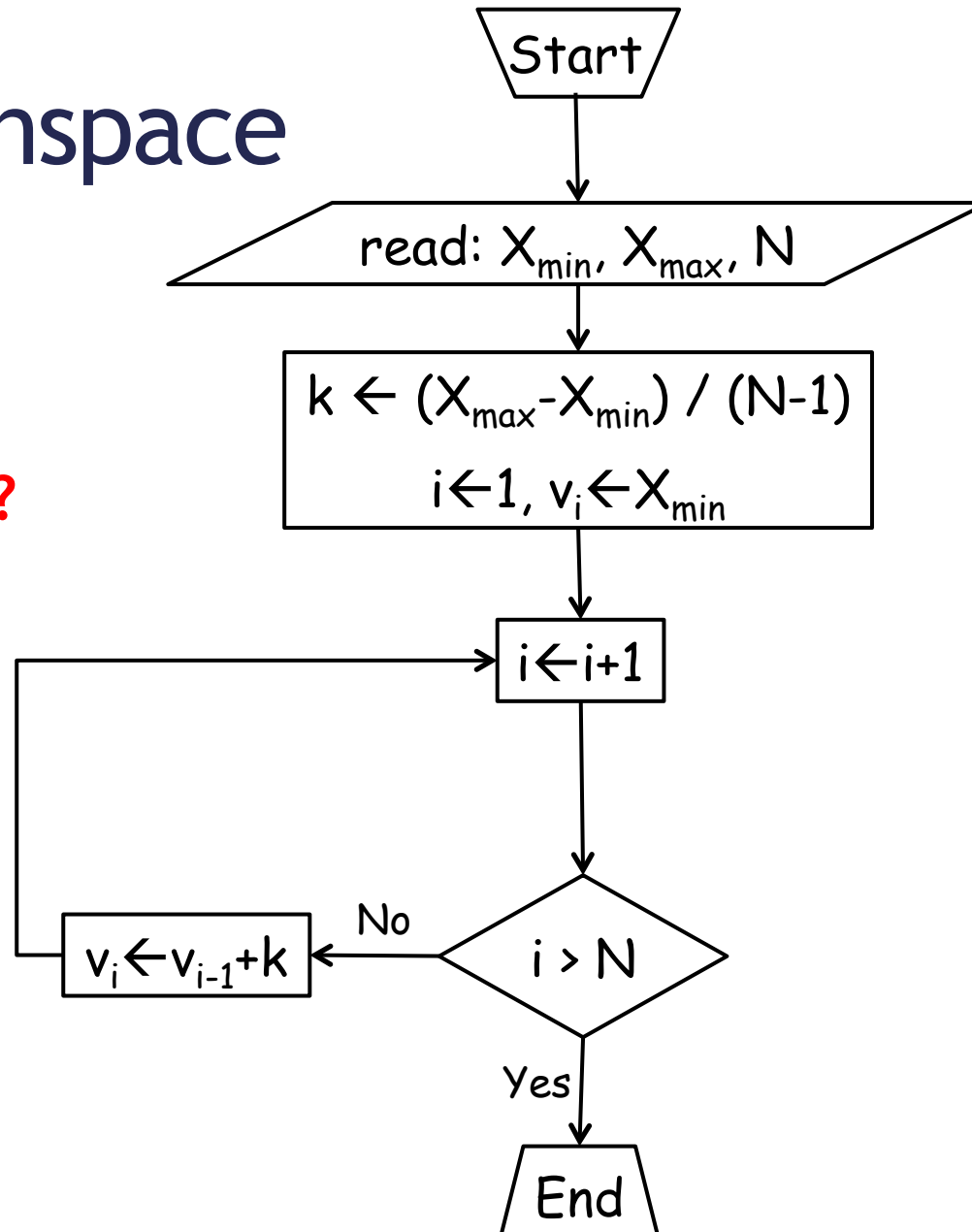
```
% Script to compute ax^2 + bx + c

disp('Evaluate quadratic ax^2+bx+c');
disp('for user inputs a, b, c, and x');
a=1; b=1; c=1; x=0;
while a~=0 || b~=0 || c~=0 || x~=0
    disp('Enter a=b=c=x=0 to terminate');
    a = input('Enter value for a: ');
    b = input('Enter value for b: ');
    c = input('Enter value for c: ');
    x = input('Enter value for x: ');
    if a==0 && b==0 && c==0 && x==0
        break
    end
    quadratic = a*x^2 + b*x + c;
    fprintf('Quadratic result: %f\n\n', quadratic);
end
```

See quadeq.m

# Example: linspace

for or while loop?



# Example: linspace with while loop

```
function vec = my_linspace(Xmin,Xmax,N)
%Computes the MATLAB linspace function
%Inputs: Xmin is the minimum value, Xmax the upper limit,
%N the number of elements
```

```
%Initialization
```

```
k=(Xmax-Xmin)/(N-1);
```

```
i=1;
```

```
vec(i)=Xmin;
```

```
%while loop to create the vector
```

```
while i<N
```

```
    %do the task
```

```
    i=i+1;
```

```
    vec(i)=vec(i-1)+k;
```

```
end
```

```
end
```

See my\_linspace.m

Calling the  
function

```
>> my_linspace(1,5,4)
```

```
ans =
```

```
    1.0000    2.3333    3.6667    5.0000
```

# Example: linspace with for loop

```
function vec = my_linspace(Xmin,Xmax,N)
%Computes the MATLAB linspace function
%Inputs: Xmin is the minimum value, Xmax the upper limit,
%N the number of elements

%Initialization
k=(Xmax-Xmin)/(N-1);
i=1;
vec(i)=Xmin;

%for loop to create the vector
for i=2:N
    %do the task
    vec(i)=vec(i-1)+k;
end
end
```

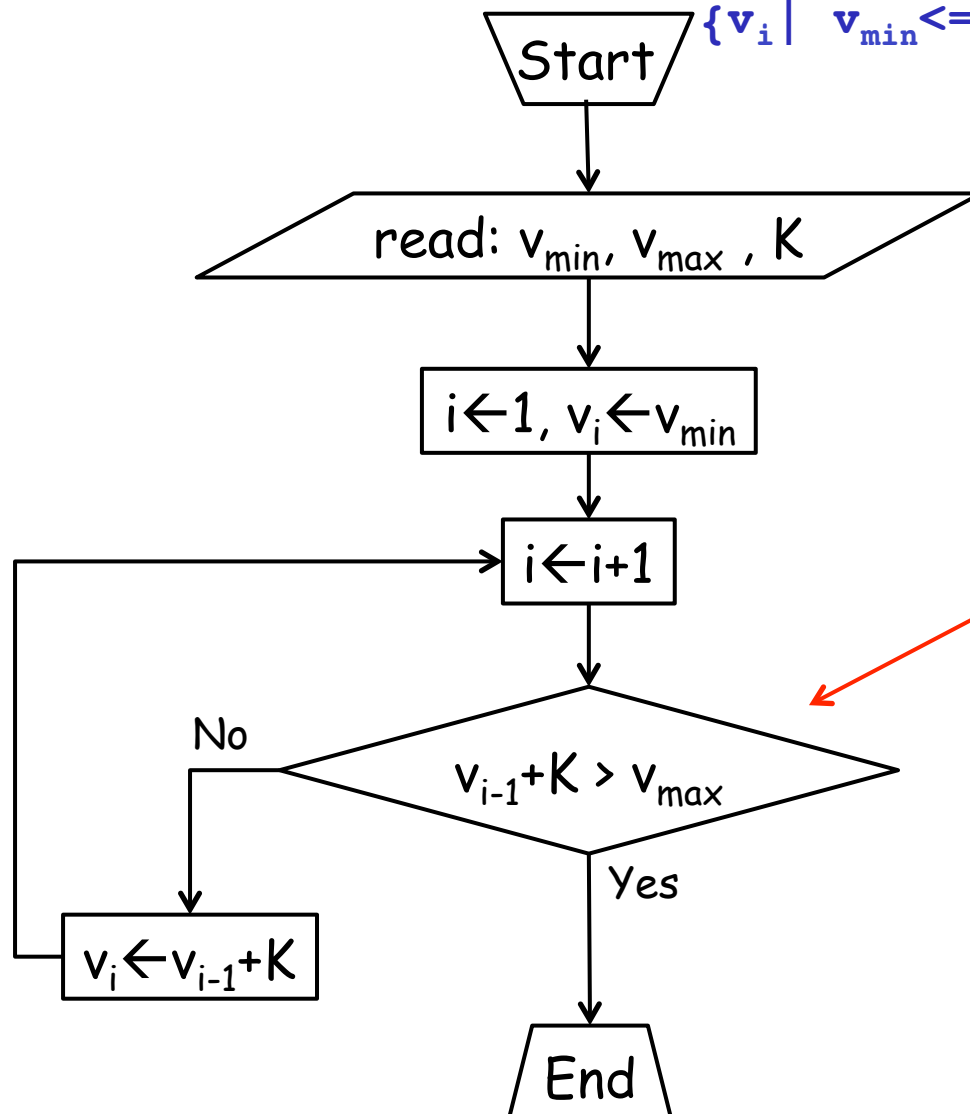
See my\_linspace.m



# Example: create a vector

$$B = [v_1, v_2, \dots, v_n] =$$

$$\{v_i \mid v_{\min} \leq v_i \leq v_{\max}, v_i = v_{i-1} + K, v_1 = v_{\min}, i = 2, \dots\}$$



**for or while loop?**

# Ex: create a vector with while loop

```
%this script creates a vector
```

```
%input  
Vmin=3;  
Vmax=15;  
K=2;
```

```
%initialization  
i=1;  
v(i)=Vmin;  
i=i+1;
```

```
%loop to create the vector  
while v(i-1)+K <= Vmax  
    %do the task  
    v(i)=v(i-1)+K;  
    i=i+1;  
end
```

```
%this script creates a vector
```

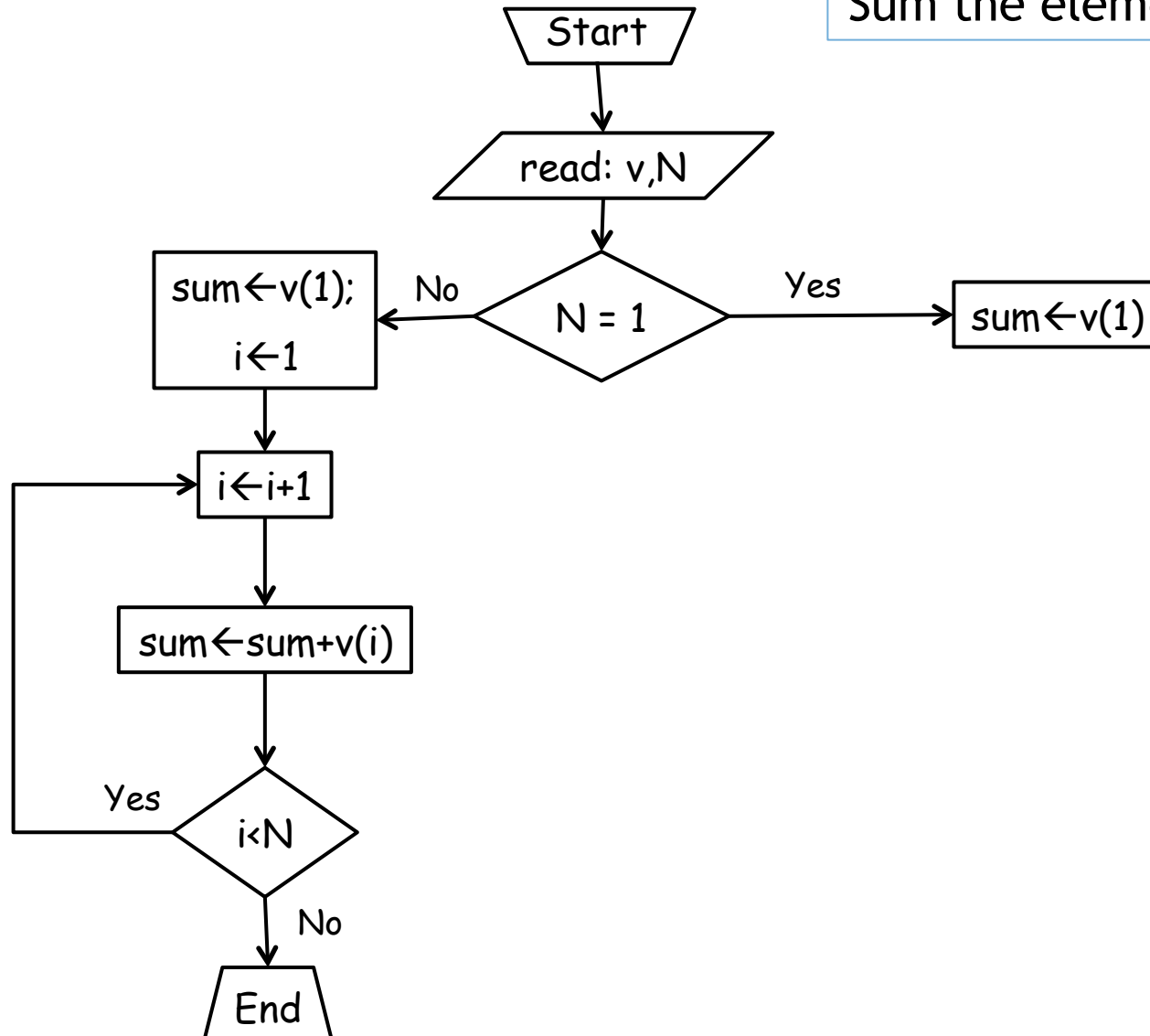
```
%input  
Vmin=3;  
Vmax=15;  
K=2;
```

```
%initialization  
i=1;  
v(i)=Vmin;
```

```
%loop to create the vector  
while v(i)+K <= Vmax  
    %do the task  
    i=i+1;  
    v(i)=v(i-1)+K;  
end
```

# mySum(v), see Lesson 5

Sum the elements of a vector



# mySum(v): multiple versions

```
%mySum(v) ver1
v=[4 5 2 19 5 7 8];
n=length(v);
if n==1
    mySum=v(1);
else
    mySum=v(1);
    for i=2:n
        mySum=mySum+v(i);
    end
end
```

```
%mySum ver2
v=[4 5 2 19 5 7 8];
mySum=v(1);
for i=2:length(v)
    mySum=mySum+v(i);
end
```

```
%mySum ver3
v=[4 5 2 19 5 7 8];
n=length(v);
mySum=v(1);
i=1;
while i<n
    i=i+1;
    mySum=mySum+v(i);
end
```

```
%mySum ver4
v=[4 5 2 19 5 7 8];
mySum=0;
for x=v
    mySum=mySum+x;
end
```

# Example: cumulative sum using for

```
function outvec = my_cumsum(invec)
% my_cumsum computes the cumulative sum of the vector invec
% Returns a vector outvec such that outvec(i) contains the sum
% of the first i elements of the vector
```

```
outvec = zeros(1, length(invec));
for i=1:length(invec)
    for j=1:i
        outvec(i) = outvec(i) + invec(j);
    end
end
end
```

Calling the  
function

```
>> A = randi([0,10],1,6)
A =
     4     10     8     10     7     0
>> my_cumsum(A)
ans =
     4    14    22    32    39    39
```

See my\_cumsum.m

# Example: cumulative sum using while

```
function outvec = my_cumsum(invec)
% my_cumsum computes the cumulative sum of the vector invec
% Returns a vector outvec such that outvec(i) contains the
% sum
% of the first i elements of the vector

%while loop
outvec = zeros(1, length(invec));
i=1;
while i<=length(invec)
    j=1;
    while j<=i
        outvec(i)=outvec(i)+invec(j);
        j=j+1;
    end
    i=i+1;
end
```

## Homework 2 (see pdf file on the course site)

Consider a row vector  $V$  of length  $N$ . Design and code:

1. A script that returns the minimum element of  $V$  and its index in the vector
2. A script that finds a value  $x$  in  $V$  and if found returns its index in the vector
3. A script that computes the mean of the elements in  $V$
4. A script that returns the minimum and maximum of the elements in  $V$  and their indexes
5. A script that computes the mean of the elements in  $V$  and returns the index of the element closest to the mean

Then consider a matrix  $A$  of size  $M \times N$

6. Solve the above problems (1-5) for matrix  $A$  (rather than vector  $V$ )