BIL 108E
Introduction to Scientific and Engineering Computing

ASST. PROF. DR. İPEK AKIN

Programming in MATLAB

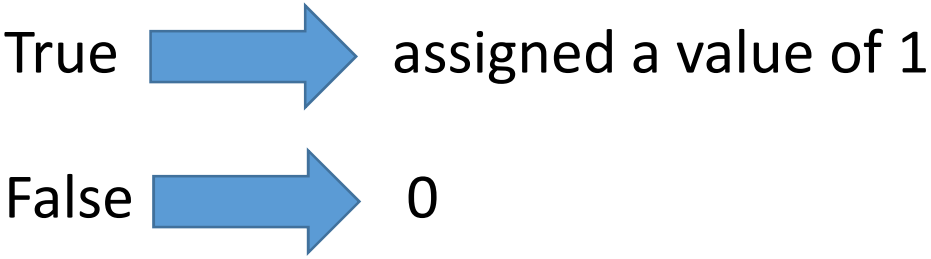
INTRODUCTION

- MATLAB provides several tools that can be used to control the flow of a program. Conditional statements and the switch structure make it possible to skip commands or to execute specific groups of commands in different situations.

- For loops and while loops make it possible to repeat a sequence of commands several times.

Relational and Logical Operators

A **relational operator** compares two numbers by determining whether a comparison statement (e.g. 5 < 8) is true or false.



A **logical operator** examines true/false statements and produces a result that is true (1) or false (0) according to the specific operator. For example, the logical AND operator gives 1 only if both statements are true.

Relational and logical operators can be used in mathematical expressions and, are frequently used in combination with other commands, to make decisions that control the flow of a computer program.

Relational Operators

Relational operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
~=	Not Equal to

Relational operators are used as arithmetic operators within a mathematical expression. The result can be used in other mathematical operations, in addressing arrays, and together with other MATLAB commands (e.g., `if`) to control the flow of a program.

When two numbers are compared, the result is 1 (logical true) if the comparison, according to the relational operator, is true, and 0 (logical false) if the comparison is false.

If two scalars are compared, the result is a scalar 1 or 0. If two arrays are compared (only arrays of the same size can be compared), the comparison is done *element-by-element*, and the result is a logical array of the same size with 1s and 0s according to the outcome of the comparison at each address.

If a scalar is compared with an array, the scalar is compared with every element of the array, and the result is a logical array with 1s and 0s according to the outcome of the comparison of each element.

Examples

Check if 5 is larger than 8.

```
>> 5>8
ans =
     0
```

Checks if 5 is larger than 8.

Since the comparison is false (5 is not larger than 8) the answer is 0.

Check if 5 is smaller than 10, assigns the answer to a.

```
>> a=5<10
a =
     1
```

Checks if 5 is smaller than 10, and assigns the answer to a.

Since the comparison is true (5 is smaller than 10) the number 1 is assigned to a.

Example (using relational operators)

```
>> y=(6<10)+(7>8)+(5*3==60/4)
```

Equal to 1 since 6 is smaller than 10.

Equal to 0 since 7 is not larger than 8.

Equal to 1 since 5*3 is equal to 60/4.

Define the vectors b and c

```
>> b=[15 6 9 4 11 7 14]; c=[8 20 9 2 19 7 10];
```

Checks which c elements are larger than or equal to b elements

```
>> d=c>=b
```

Checks which c elements are larger than or equal to b elements.

```
d =
     0     1     1     0     1     1     0
```

Assigns 1 where an element of c is larger than or equal to an element of b.

Define the vectors b and c

```
>> b=[15 6 9 4 11 7 14]; c=[8 20 9 2 19 7 10];
```

Checks which b elements are equal to c elements

```
>> b == c                    Checks which b elements are equal to c elements.
ans =
     0     0     1     0     0     1     0
```

Checks which b elements are not equal to c elements

```
>> b~=c                      Checks which b elements are not equal to c elements.
ans =
     1     1     0     1     1     0     1
```

Subtracts c from b and then checks which elements are larger than 0

```
>> f=b-c>0                   Subtracts c from b and then checks
f =                          which elements are larger than zero.
     1     0     0     1     0     0     1
```

Define a 3 x 3 matrix A

```
>> A=[2 9 4; -3 5 2; 6 7 -1]
```

Checks which elements in A are smaller than or equal to 2. Assigns the results to matrix B.

```
>> B=A<=2
```

```
B =

     1     0     0
     1     0     1
     0     0     1
```

- The results of a relational operation with vectors, which are vectors with 0s and 1s, are called **logical vectors** and can be used **for addressing vectors**.

- **When a logical vector is used for addressing another vector, it extracts from that vector the elements in the positions where the logical vector has 1s**

Define a vector r

```
>> r = [8 12 9 4 23 19 10]
```

Checks which r elements are smaller than or equal to 10.

```
>> s=r<=10
```

```
s =

     1     0     1     1     0     0     1
```

Use s for addresses in vector r to create vector t

```
>> t=r(s)
t =
     8     9     4    10
>> w=r(r<=10)
w =
     8     9     4    10
```

Use s for addresses in vector r to create vector t.

Vector t consists of elements of r in positions where s has 1s.

The same procedure can be done in one step.

- Numerical vectors and arrays with the numbers 0s and 1s are not the same as logical vectors and arrays with 0s and 1s.
- **Numerical vectors and arrays can not be used for addressing**.
- Logical vectors and arrays, however, can be used in arithmetic operations.
- The first time a logical vector or an array is used in arithmetic operations it is changed to a numerical vector or array.

Order of Precedence

The relational operators themselves have equal precedence and are evaluated from left to right. Parentheses can be used to alter the order of precedence.

Order of Precedence

```
>> 3+4<16/2
ans =
     1
```

+ and / are executed first.

The answer is 1 since 7 < 8 is true.

```
>> 3+(4<16)/2
ans =
    3.5000
```

4 < 16 is executed first, and is equal to 1, since it is true.

3.5 is obtained from 3 + 1/2.

Logical Operators

Logical operator	**Name**	**Description**
& Example: A&B	AND	Operates on two operands (A and B). If both are true, the result is true (1); otherwise the result is false (0).
| Example: A|B	OR	Operates on two operands (A and B). If either one, or both, are true, the result is true (1); otherwise (both are false) the result is false (0).
~ Example: ~A	NOT	Operates on one operand (A). Gives the opposite of the operand; true (1) if the operand is false, and false (0) if the operand is true.

Logical Operators

- Logical operators have numbers as operands. A nonzero number is true, and a zero number is false.
- Logical operators (like relational operators) are used as arithmetic operators within a mathematical expression.
- The result can be used in other mathematical operations, in addressing arrays, and together with other MATLAB commands (e.g., if) to control the flow of a program.
- Logical operators (like relational operators) can be used with scalars and arrays.

Logical Operators

- The logical operations AND and OR can have both operands as scalars, arrays, or one array and one scalar.
- If both are scalars, the result is a scalar 0 or 1. If both are arrays, they must be of the same size and the logical operation is done *element-by-element*.
- The result is an array of the same size with 1s and 0s according to the outcome of the operation at each position.
- If one operand is a scalar and the other is an array, the logical operation is done between the scalar and each of the elements in the array and the outcome is an array of the same size with 1s and 0s.

- The logical operation NOT has one operand. When it is used with a scalar the outcome is a scalar 0 or 1.

- When it is used with an array, the outcome is an array of the same size with 1s in positions where the array has nonzero numbers and 0s in positions where the array has 0s.

Examples

```
>> 3&7
ans=
     1
```

& Example: A&B	AND	Operates on two operands (A and B). If both are true, the result is true (1); otherwise the result is false (0).

```
>> a=5|0
a =
     1
>> ~25
ans =
     0
```

5 OR 0 (assign to variable a).

1 is assigned to a since at least one number is true (nonzero).

NOT 25.

The outcome is 0 since 25 is true (nonzero) and the opposite is false.

```
>> t=25*((12&0)+(~0)+(0|5))
t =
    50
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];
>> x&y
ans =
     1     0     0     1     0     1
```

Using logical operators in a math expression.

Define two vectors x and y.

The outcome is a vector with 1 in every position where both x and y are true (nonzero elements), and 0s otherwise.

```
>> z=x|y
z =
     1     1     1     1     0     1
```

The outcome is a vector with 1 in every position where either or both x and y are true (nonzero elements), and 0s otherwise.

```
>> ~(x+y)
ans =
     0     0     0     1     1     0
```

The outcome is a vector with 0 in every position where the vector x + y is true (nonzero elements), and 1 in every position where x + y is false (zero elements).

Order of Precedence

Precedence	Operation
1 (highest)	Parentheses (if nested parentheses exist, inner ones have precedence)
2	Exponentiation
3	Logical NOT (~)
4	Multiplication, division
5	Addition, subtraction
6	Relational operators (>, <, >=, <=, ==, ~=)
7	Logical AND (&)
8 (lowest)	Logical OR (|)

- If two or more operations have the same precedence, the expression is executed in order from left to right.

Built-In Logical Functions

`and(A,B)`	equivalent to `A&B`
`or(A,B)`	equivalent to `A|B`
`not(A)`	equivalent to `~A`

Built-In Logical Functions

Function	Description	Example

`xor(a,b)`	Exclusive or. Returns true (1) if one operand is true and the other is false.	`>> xor(7,0)`
		`ans =`
		`1`
		`>> xor(7,-5)`
		`ans =`
		`0`

`all(A)`	Returns 1 (true) if all elements in a vector `A` are true (nonzero). Returns 0 (false) if one or more elements are false (zero).
	If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.
		`>> A=[6 2 15 9 7 11];`
		`>> all(A)`
		`ans =`
		`1`
		`>> B=[6 2 15 9 0 11];`
		`>> all(B)`
		`ans =`
		`0`

`any(A)`	Returns 1 (true) if any element in a vector `A` is true (nonzero). Returns 0 (false) if all elements are false (zero).
	If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.
		`>> A=[6 0 15 0 0 11];`
		`>> any(A)`
		`ans =`
		`1`
		`>> B = [0 0 0 0 0 0];`
		`>> any(B)`
		`ans =`
		`0`

`find(A)`
 	
`find(A>d)`
	If `A` is a vector, returns the indices of the nonzero elements.
	If `A` is a vector, returns the address of the elements that are larger than `d` (any relational operator can be used).
		`>> A=[0 9 4 3 7 0 0 1 8];`
		`>> find(A)`
		`ans =`
		`2 3 4 5 8 9`
		`>> find(A>4)`
		`ans =`
		`2 5 9`

The operations of the four logical operators, and, or, xor, and not can be summarized in a truth table:

INPUT		OUTPUT				
A	B	AND A&B	OR A|B	XOR (A,B)	NOT ~A	NOT ~B
false	false	false	false	false	true	true
false	true	false	true	true	true	false
true	false	false	true	true	false	true
true	true	true	true	false	false	false

Conditional Statements

If the expression is true, a group of commands that follow the statement are executed. If the expression is false, the computer skips the group.

```
if conditional expression consisting of relational and/or logical operators.
```

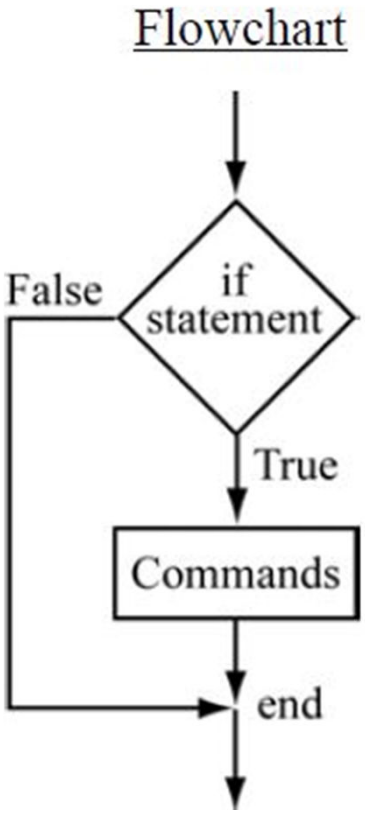
```
if a < b
if c >= 5
if a == b
if a ~= 0
if (d<h)&(x>7)
if (x~=13)|(y<0)
```

All the variables must have assigned values.

if-end Structure

If the conditional expression is true (1), the program continues to execute the commands that follow the if statement all the way down to the end statement.

If the conditional expression is false (0), the program skips the group of commands btw the if and the end, and continues with the commands that follow the end.

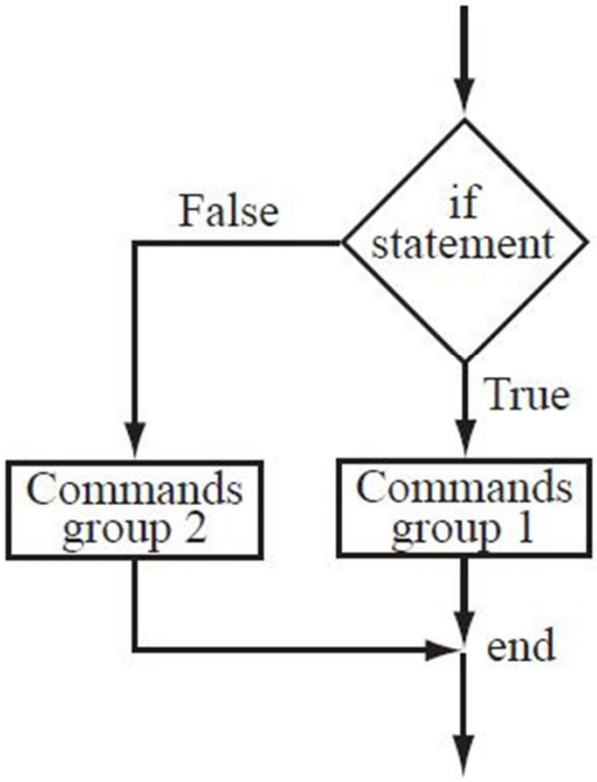



```
.....
.....  MATLAB program.
.....
if  conditional expression
        ........
        ........      A group of
        ........      MATLAB commands
end
.....
.....  MATLAB program.
.....
```

if-else-end Structure

Provides a means for choosing one group of commands, out of a possible 2 groups, for execution.

Flowchart



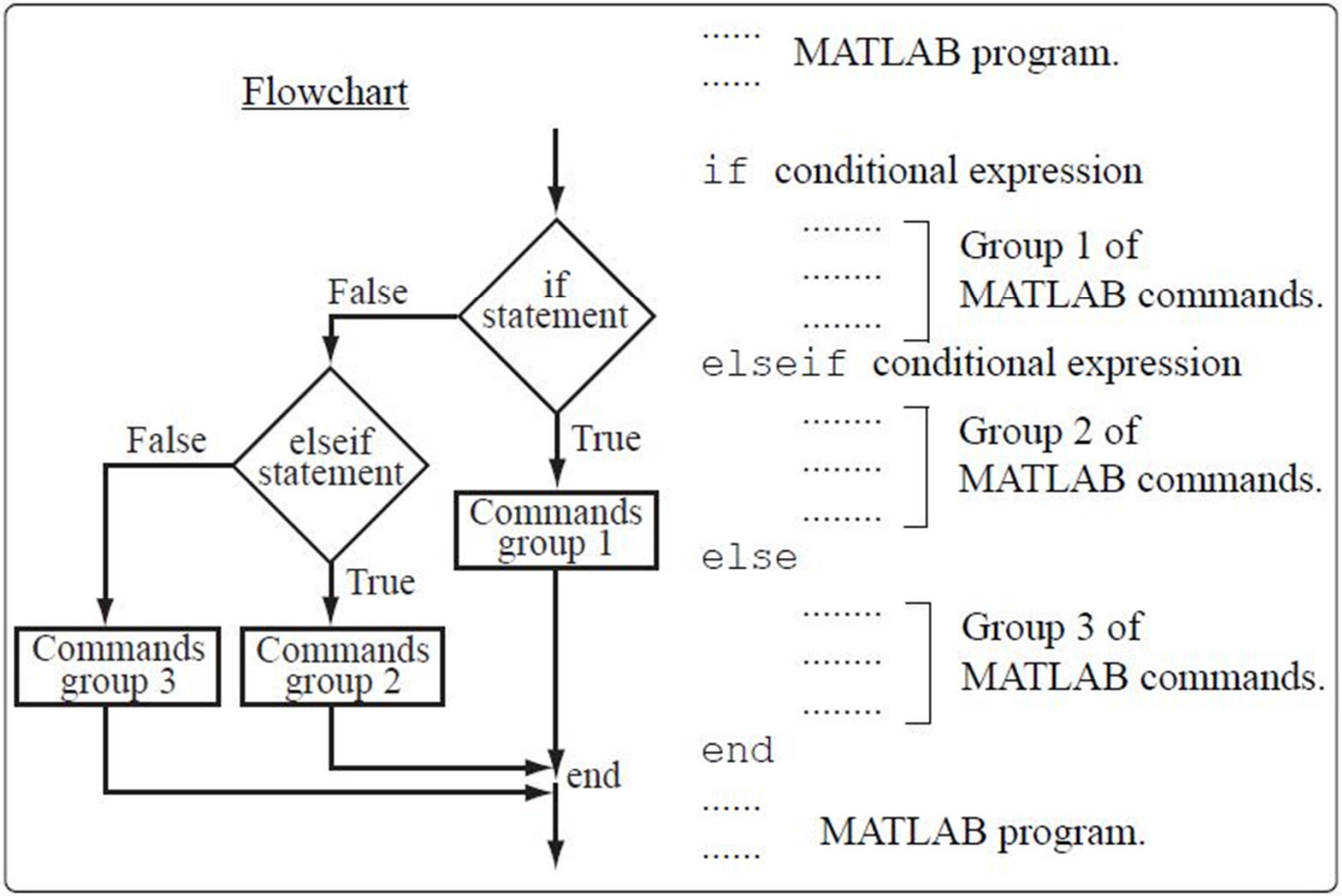

```
.....
.....   MATLAB program.

if conditional expression
        .....
        .....   Group 1 of
        .....   MATLAB commands
else
        .....
        .....   Group 2 of
        .....   MATLAB commands
end
.....
.....   MATLAB program.
```

If the conditional expression is true, the program executes group 1 of commands between the if and the else statements and then skips to the end. If the conditional expression is false, the program skips to the else and then executes group 2 of commands between the else and the end.

if-elseif-else-end Structure

Possible to select one out of 3 groups of commands for execution



switch-case Statement

Another method that can be used to direct the flow of a program. It provides a means for choosing one group of commands for execution out of several possible groups.

```
switch  switch expression
```

```
.....
.....      MATLAB program.

switch switch expression
    case value1
    .....
    .....      ] Group 1 of commands.
    case value2
    .....
    .....      ] Group 2 of commands.
    case value3
    .....
    .....      ] Group 3 of commands.
    otherwise
    .....
    .....      ] Group 4 of commands.
end
.....
.....      MATLAB program.
.....
```

Loops

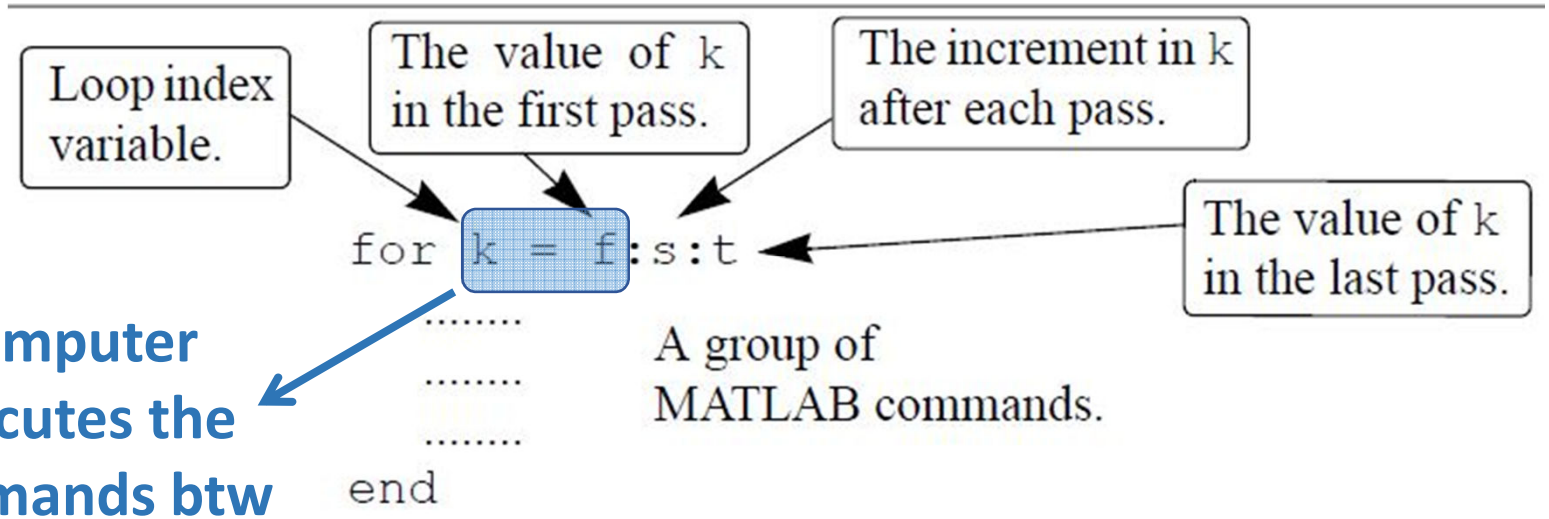
- A loop is another method to alter the flow of a computer program. In a loop, the execution of a command, or a group of commands, is repeated several times consecutively.

- Each round of execution is called a pass. In each pass at least one variable, but usually more than one, or even all the variables that are defined within the loop, are assigned new values.
    - for-end loops
    - while-end loops

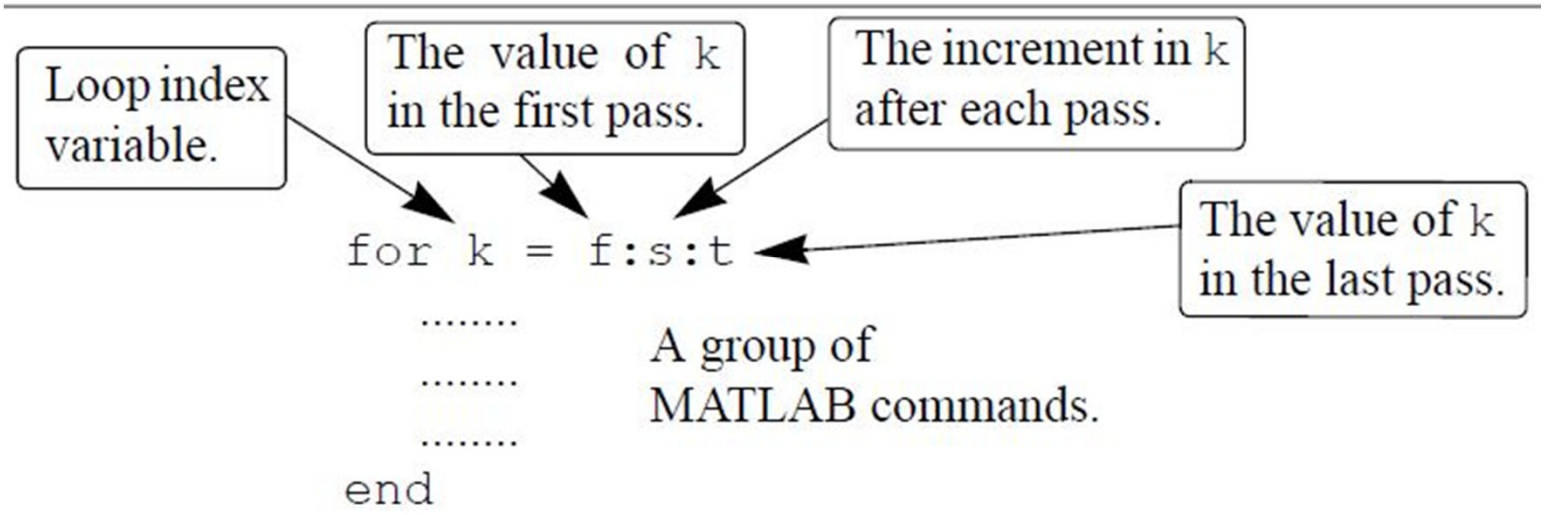
for-end Loop

The execution of a command, or a group of commands, is repeated a predetermined number of times.

The loop index variable can have any variable name (usually i, j, k, m, and n are used, however, i and j should not be used if MATLAB is used with complex numbers.



computer executes the commands btw the for and end commands



- Then, the program goes back to the for command for the second pass. k obtains a new value equal to k = f + s, and the commands btw the for and end commands are executed with the new value of k.

- The process repeats itself until the last pass, where k = t. Then the program does not go back to the for, but continues with the commands that follow the end command.

- For example, if k = 1:2:9, there are five loops, and the corresponding values of k are 1, 3, 5, 7, and 9.

- The increment s can be negative (i.e.; k = 25:–5:10 produces four passes with k = 25, 20, 15, 10).

    - If the increment value s is omitted, the value is 1 (default) (i.e.; k = 3:7 produces five passes with k = 3, 4, 5, 6, 7).

If `f` = `t`, the loop is executed once.

If `f` > `t` and `s` > 0, or if `f` < `t` and `s` < 0, the loop is not executed.

If the values of `k`, `s`, and `t` are such that `k` cannot be equal to `t`, then if `s` is positive, the last pass is the one where `k` has the largest value that is smaller than `t` (i.e., `k` = 8:10:50 produces five passes with `k` = 8, 18, 28, 38, 48). If `s` is negative, the last pass is the one where `k` has the smallest value that is larger than `t`.

In the `for` command `k` can also be assigned a specific value (typed as a vector). Example: `for k = [7 9 -1 3 3 5].`

The value of `k` should not be redefined within the loop.

Each `for` command in a program *must* have an `end` command.

The value of the loop index variable (`k`) is not displayed automatically. It is possible to display the value in each pass (which is sometimes useful for debugging) by typing `k` as one of the commands in the loop.

A simple example of a `for-end` loop (in a script file) is:

```
for k=1:3:10
     x = k^2
end
```

while-end Loop

- Used in situations when looping is needed but the number of passes is not known in advance.

- In while-end loops the number of passes is not specified when the looping process starts. Instead, the looping process continues until a stated condition is satisfied.

```
while conditional expression
    ........
    ........          A group of
    ........          MATLAB commands
end
```

- The first line is a while statement that includes a conditional expression.
- When the program reaches this line the conditional expression is checked. If it is false (0), MATLAB skips to the end statement and continues with the program. If the conditional expression is true (1), MATLAB executes the group of commands that follow between the while and end commands.
- Then MATLAB jumps back to the while command and checks the conditional expression. This looping process continues until the conditional expression is false.

```
x=1
while x<=15
    x=2*x
end
```

Initial value of x is 1.

The next command is executed only if x <= 15.

In each pass x doubles.

```
x =
     1
x =
     2
x =
     4
x =
     8
x =
    16
```

Initial value of x.

In each pass x doubles.

When x = 16, the conditional expression in the `while` command is false and the looping stops.

break Command

- When inside a loop (for or while), the break command terminates the execution of the loop (the whole loop, not just the last pass).
- When the break command appears in a loop, MATLAB jumps to the end command of the loop and continues with the next command (it does not go back to the for command of that loop).

continue Command

- The continue command can be used inside a loop (for or while) to stop the present pass and start the next pass in the looping process