Chapter 6

Programming in MATLAB

MATLAB An Introduction With Applications, 6th Edition
Dr. Amos Gilat
The Ohio State University

Slide deck by
Dr. Greg Reese
Miami University

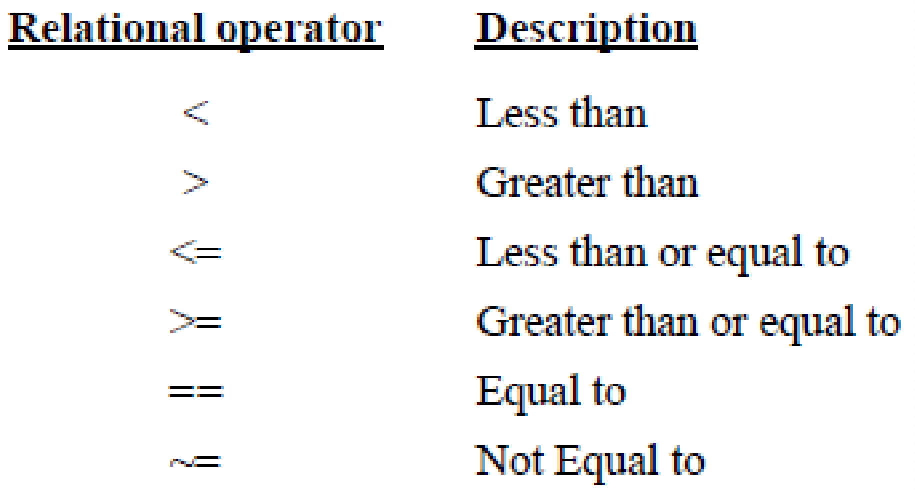
In this chapter will study how to make MATLAB programs run sections of code

- <u>If</u> something is true
- <u>While</u> something is true
- <u>For</u> a certain number of times

Will also learn how to run different sections of code depending on

- The value of a variable
- Which particular condition is true
- What combination of conditions is true
  - If this <u>and</u> that are true
  - If this <u>or</u> that is true, etc.
- What relationship two things have
  - For example, one is less than the other; greater than; equal to; not equal to; etc.

Relational operator:



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

- Can't put space between operators that have two characters
- "Not equal to" is "~=", not "!=" as in C or C++
- "Equal to" comparison is two equal signs (==), not one.
  - Remember, "=" means "assign to" or "put into"

6.1 RELATIONAL AND LOGICAL OPERATORS

- Result of comparing with a relational operator is always "true" or "false"
  - If "true", MATLAB gives the comparison a value of one (1)
  - If "false", MATLAB gives the comparison a value of zero (0)

This may be different than convention in other programming languages. For example, C gives an expression that is false a value of zero, but it can give a true expression any value but zero, which you can't assume will be one

When comparing arrays

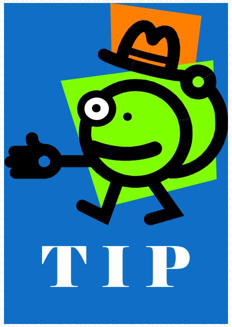
- They must be the same dimensions
- MATLAB does an elementwise comparison
- Result is an array that has same dimensions as other two but only contains 1's and 0's

When comparing array to scalar

- MATLAB compares scalar to every member of array
- Result is an array that has same dimensions as original but only contains 1's and 0's

Example

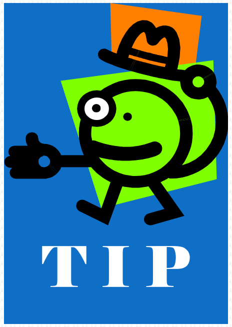
```
>> x=8:12
x =       8     9    10    11    12
>> x>10
ans =     0     0     0     1     1
>> x==11
ans =     0     0     0     1     0
>> x>=7
ans =     1     1     1     1     1
```



It helps to picture in your mind that the result of a logical comparison

1. Is a vector
2. Has a 0 or 1 corresponding to each original element

```
>> x=8:12
x =         8       9      10      11      12
>> x>10
ans =       0       0       0       1       1
```



If results of relational comparison stored in a vector, can easily find the number of elements that satisfy that comparison, i.e., that are true, by using `sum` command, which returns sum of vector elements

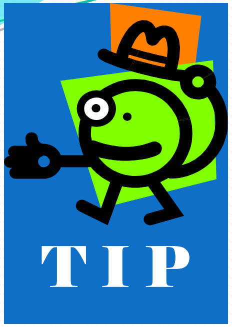
- Works because elements that are true have value of one and false elements have value zero

EXAMPLE

How many of the numbers from 1-20 are prime?

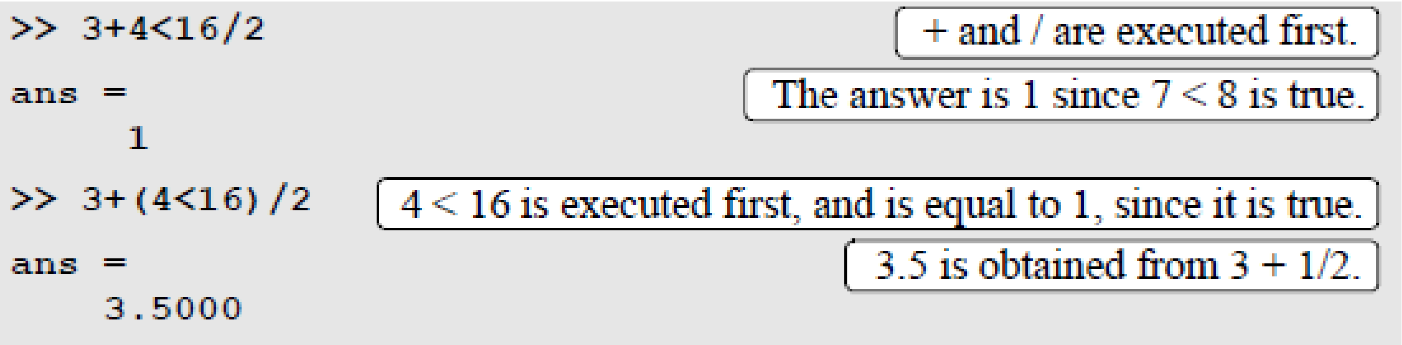
- Use MATLAB `isprime` command, which returns true (1) is number is prime and false (0) if it isn't

```
>> numbers = 1:20;
>> sum( isprime(numbers) )
ans =
     8
```



Can mix relational and arithmetic operations in one expression

- Arithmetic operations follow usual precedence and always have higher precedence than relational operations
- Relational operations all have equal precedence and evaluated left to right



```
>> 3+4<16/2

ans =
     1

>> 3+(4<16)/2

ans =
    3.5000
```

+ and / are executed first.

The answer is 1 since 7 < 8 is true.

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

3.5 is obtained from 3 + 1/2.

A *logical vector* or *logical array* is a vector/array that has only logical 1's and 0's

- 1's and 0's from mathematical operations don't count
- 1's and 0's from relational comparisons do work
- First time a logical vector/array used in arithmetic, MATLAB changes it to a numerical vector/array

Can use logical vector to get actual values that satisfy relation, not just whether or not relation satisfied. Doing this is called *logical indexing* or *logical subscripting*

- Do this by using logical vector as index in vector of values. Result is values that satisfy relation, i.e., values for which relationship are 1
- NOTE – technique doesn't quite work with arrays. Won't discuss that case further

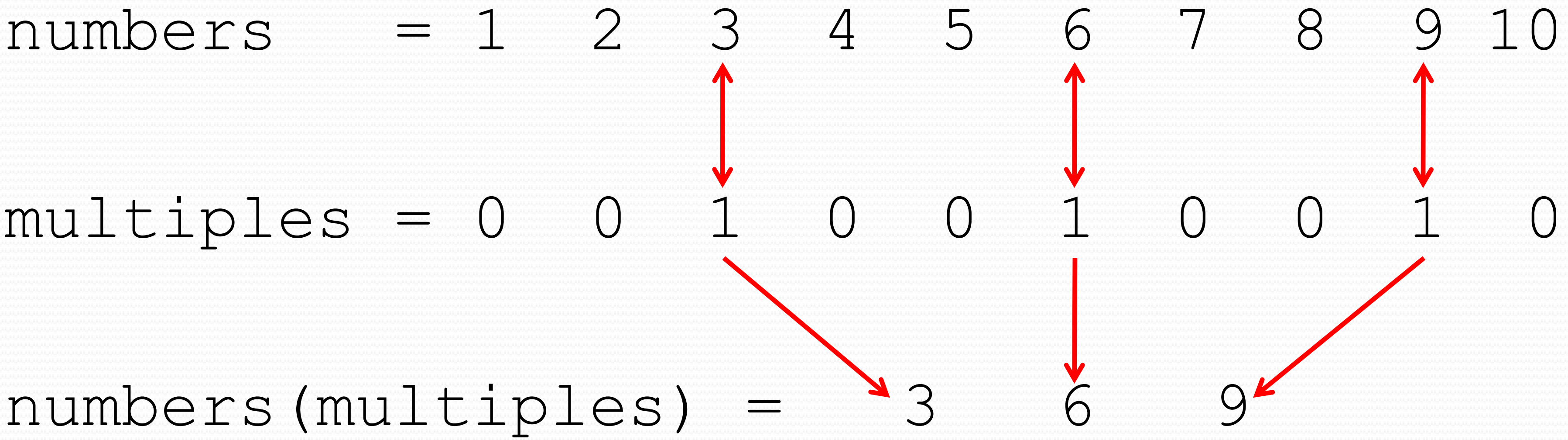
EXAMPLE

What are the numbers from 1-10 that are multiples of 3?

```
>> numbers = 1:10
numbers    = 1   2   3   4   5   6   7   8   9
   10
>> multiples = rem( numbers, 3 ) == 0
multiples = 0   0   1   0   0   1   0   0   1
   0
>> multiplesOf3 = numbers(multiples)
multiplesOf3 =
      3   6   9
```

Example

Think of `numbers(multiples)` as pulling out of `numbers` all elements that have a `1` in the corresponding element of `multiples`



```
numbers    = 1   2   3   4   5   6   7   8   9 10

multiples  = 0   0   1   0   0   1   0   0   1   0

numbers(multiples)  =    3    6    9
```

EXAMPLE

What are the prime numbers from 1-20?

```
>> numbers = 1:20;
>> numbers( isprime(numbers) )
ans =
     2   3   5   7   11   13   17   19
```

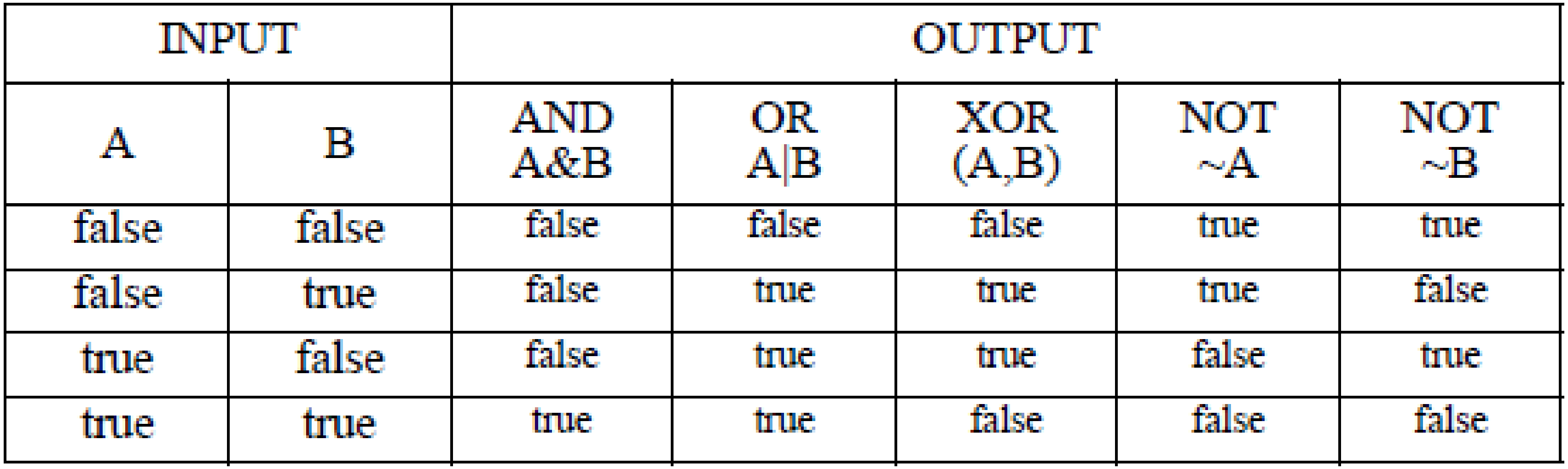
Logical indexing is particularly useful when used with logical operators, discussed next

Logical operators:

Boolean logic is a system for combining expressions that are either true of false.

- MATLAB has operators and commands to do many Boolean operations
- Boolean operations in combination with relational commands let you perform certain types of computations clearly and efficiently

A *truth table* defines the laws of Boolean logic. It gives the output of a logical operation for every possible combination of inputs. The truth table relevant to MATLAB is



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

In words, the truth table says

- AND is true if both inputs are true, otherwise it is false
- OR is true if at least one input is true, otherwise it is false
- XOR (exclusive OR) is true if exactly one input is true, otherwise it is false
- NOT is true if the input is false, otherwise it is false

An arithmetic operator, e.g., + or -, is a symbol that causes MATLAB to perform an arithmetical operation using the numbers or expressions on either side of the symbol

Similarly, a *logical operator* is a character that makes MATLAB perform a logical operation on one or two numbers or expressions

MATLAB has three logical operators: `&`, `|`, `~`

- `a&b` does the logical AND operation on `a` and `b`
- `a|b` does the logical OR operation on `a` or `b`
- `~a` does the logical NOT operation on `a`
- Arguments to all logical operators are numbers
  - Zero is "false"
  - Any non-zero number is "true"
- Result (output) of logical operator is a logical one (true) or zero (false)

When using logical operator on arrays

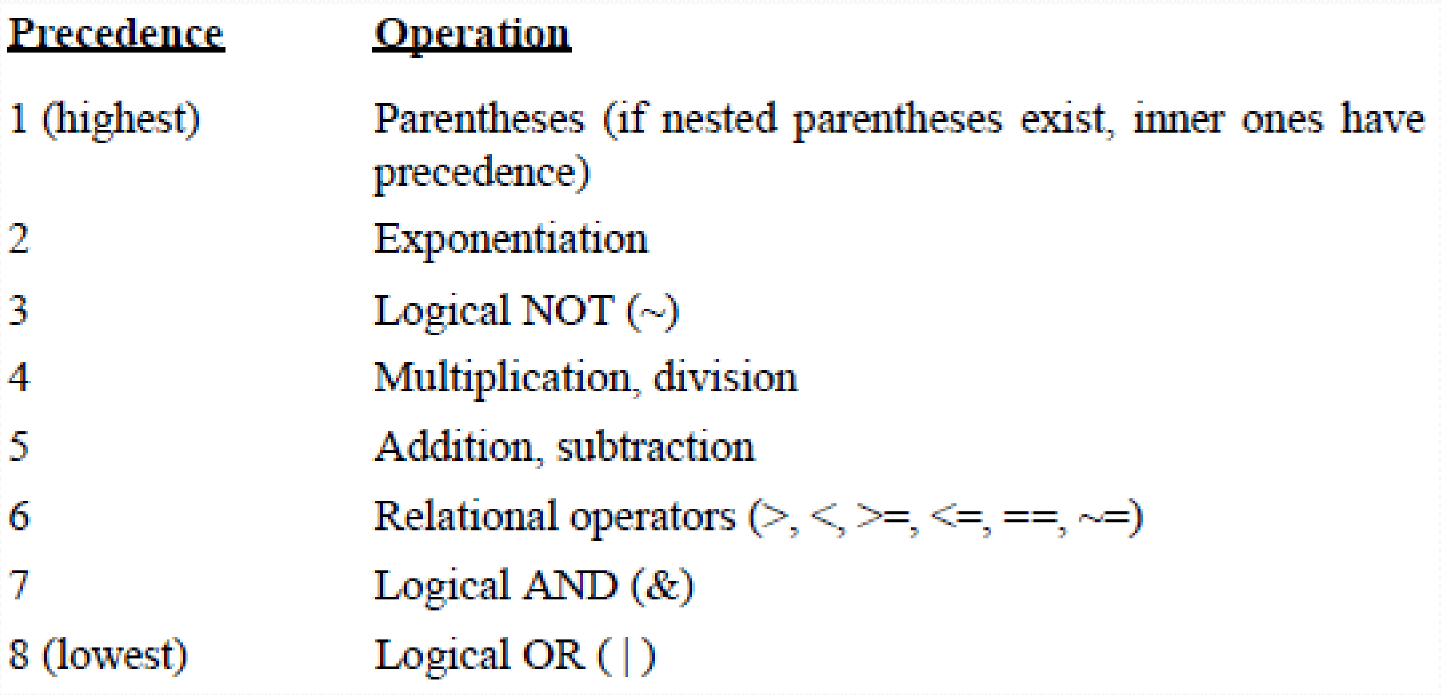
- They must be the same dimensions
- MATLAB does an element-wise evaluation of operator
- Result is an array that has same dimensions as other two but only contains 1's and 0's

(`not` only operates on one array so the first point is irrelevant)

When operating with array and scalar

- MATLAB does element-wise operation on each array element with scalar
- Result is an array that has same dimensions as original but only contains 1's and 0's

Can combine arithmetic, relational operators, and logical operators. Order of precedence is



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 (|)

EXAMPLE

Child – 12 or less years

Teenager – more than 12 and less than 20 years

Adult – 20 or more years

```
>> age=[45 47 15 13 11]
age =   45      47      15      13      11
```

EXAMPLE

Who is a teenager?

```
>> age=[45 47 15 13 11];
>> age>=13
ans =      1     1     1     1     0
>> age<=19
ans =      0     0     1     1     1
>> age>=13 & age<=19
ans =      0     0     1     1     0
```

These mark the two teenagers

EXAMPLE

```
>> age=[45 47 15 13 11]
age =   45     47     15     13     11
```

Who is not a teenager?

```
>> ~(age>=13 & age<=19)
ans =    1      1      0      0      1
```

Who is an adult or a child?

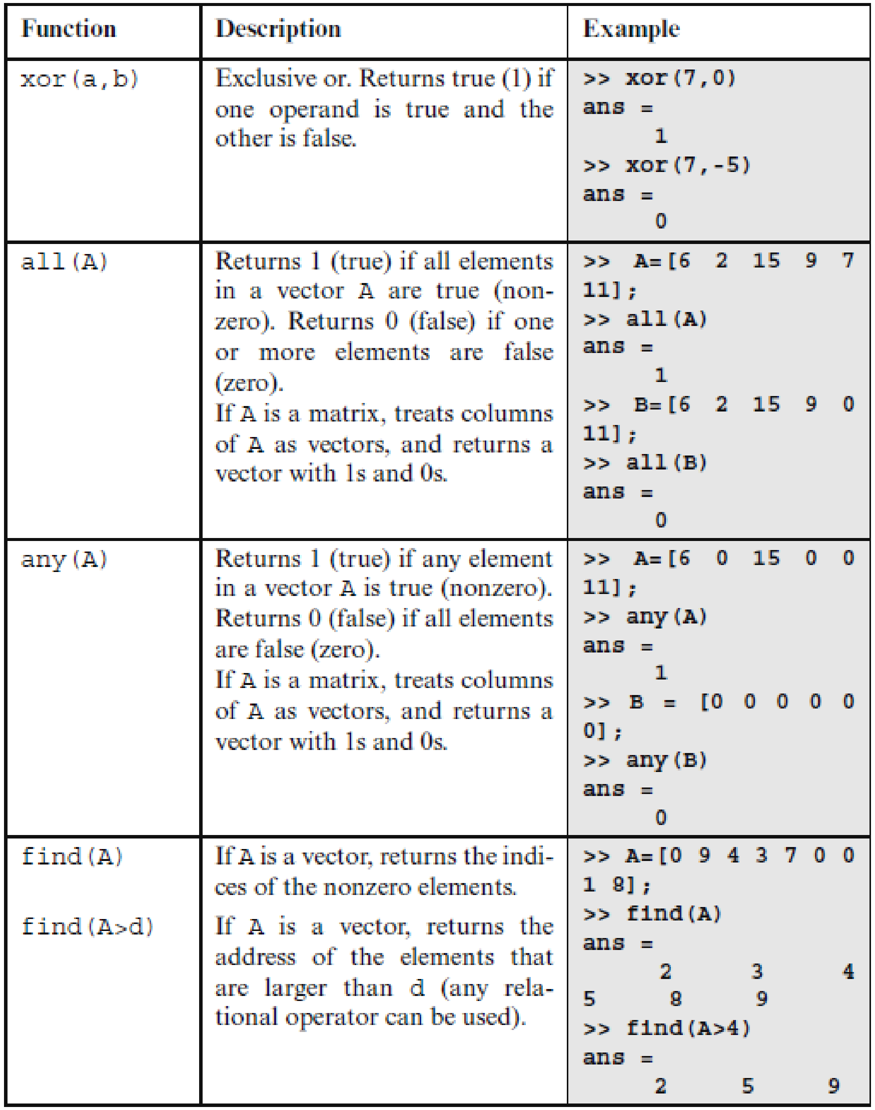
```
>> age>19 | age<13
ans =    1      1      0      0      1
```

Built-in logical functions:

MATLAB has some built-in functions or commands for doing logical operations and related calculations. Three are equivalent to the logical operators

- `and(A,B)` – same as `A&B`
- `or(A,B)` – same as `A|B`
- `not(A)` – same as `~A`

MATLAB also has other Boolean 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`

A *conditional statement* is a command that allows MATLAB to decide whether or not to execute some code that follows the statement

- Conditional statements almost always part of scripts or functions
- They have three general forms
  - `if-end`
  - `if-else-end`
  - `if-elseif-else-end`

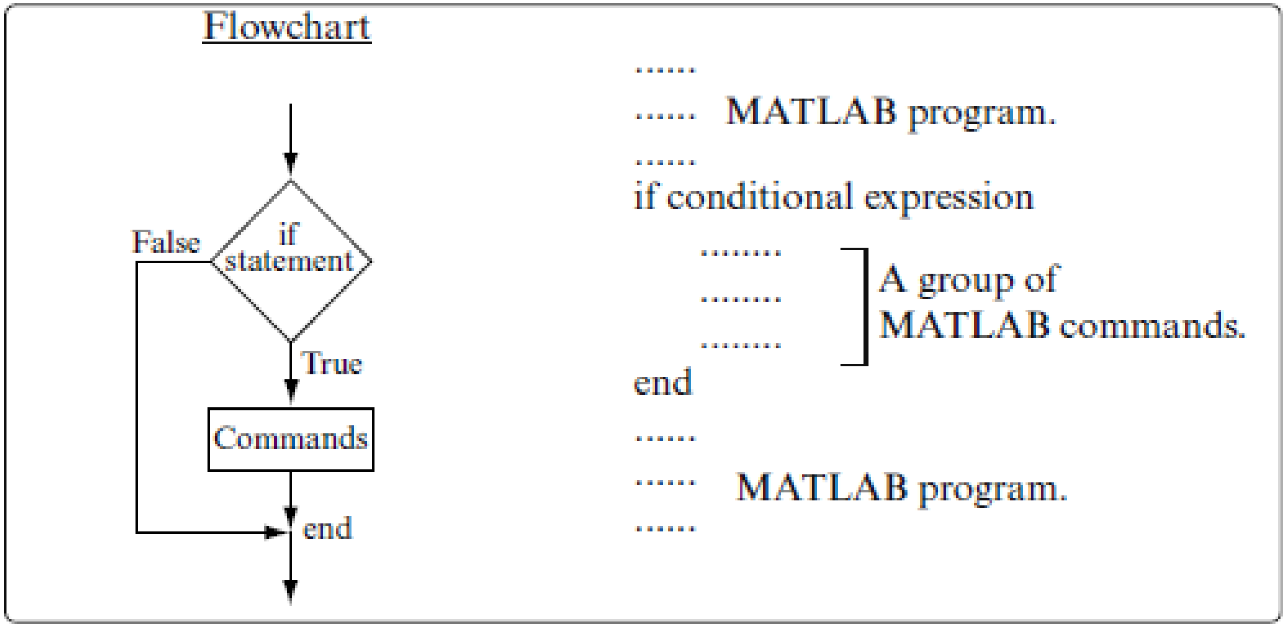
6.2.1 The `if-end` Structure

A *flowchart* is a diagram that shows the code flow. It is particularly useful for showing how conditional statements work. Some common flowchart symbols are

- ▭ represents a sequence of commands
- ◇ represents an if-statement
- → shows the direction of code execution

6.2.1 The `if-end` Structure

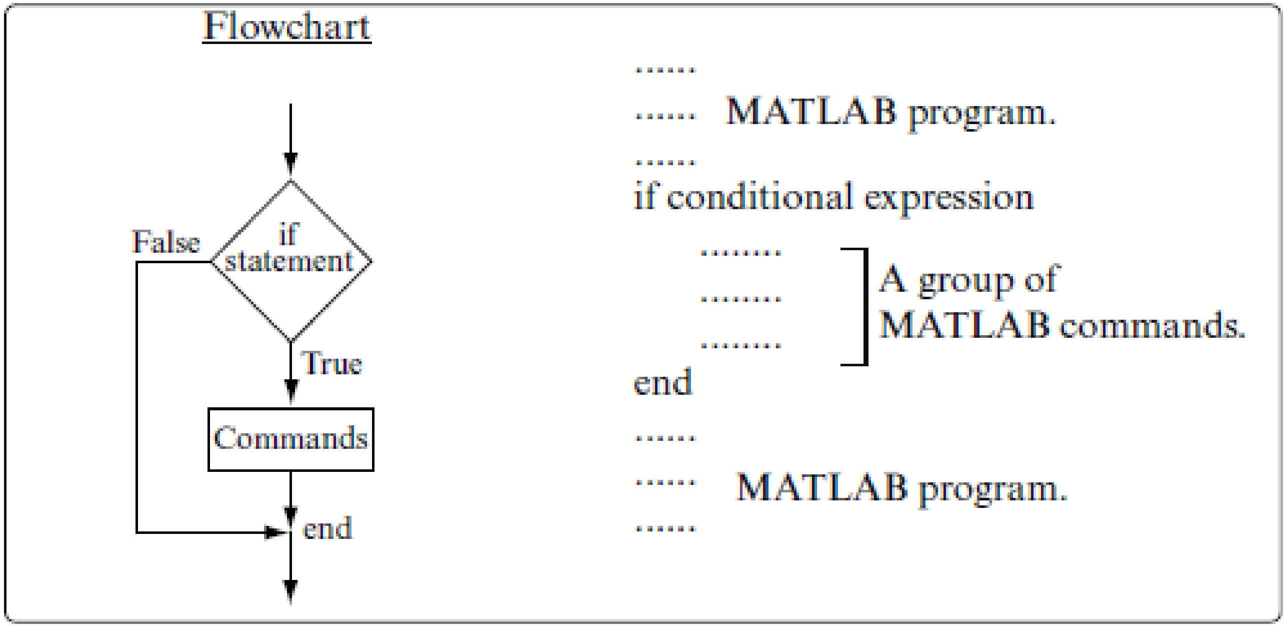
If the conditional expression is true, MATLAB runs the lines of code that are between the line with `if` and the line with `end`. Then it continues with the code after the `end`-line



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

6.2.1 The `if-end` Structure

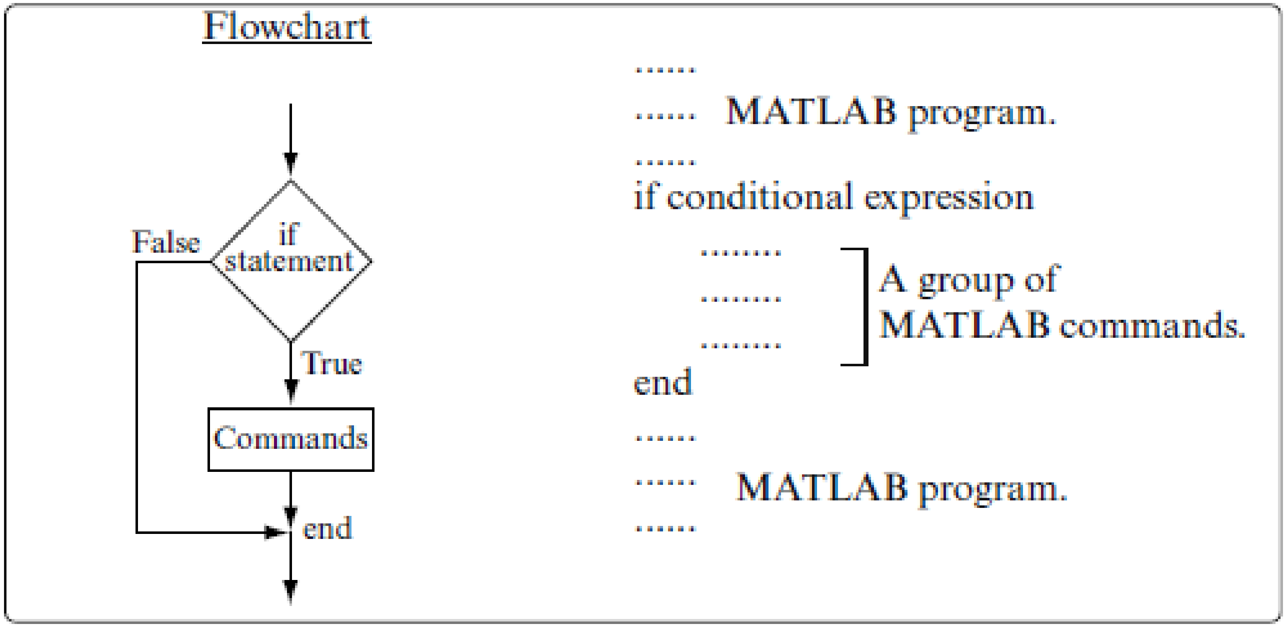
If the conditional expression is false, MATLAB skips the lines of code that are between the line with `if` and the line with `end`. Then it continues with the code after the `end`-line



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

6.2.1 The `if-end` Structure

The conditional expression is true if it evaluates to a logical 1 or to a non-zero number. The conditional expression is false if it evaluates to a logical 0 or to a numerical zero



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

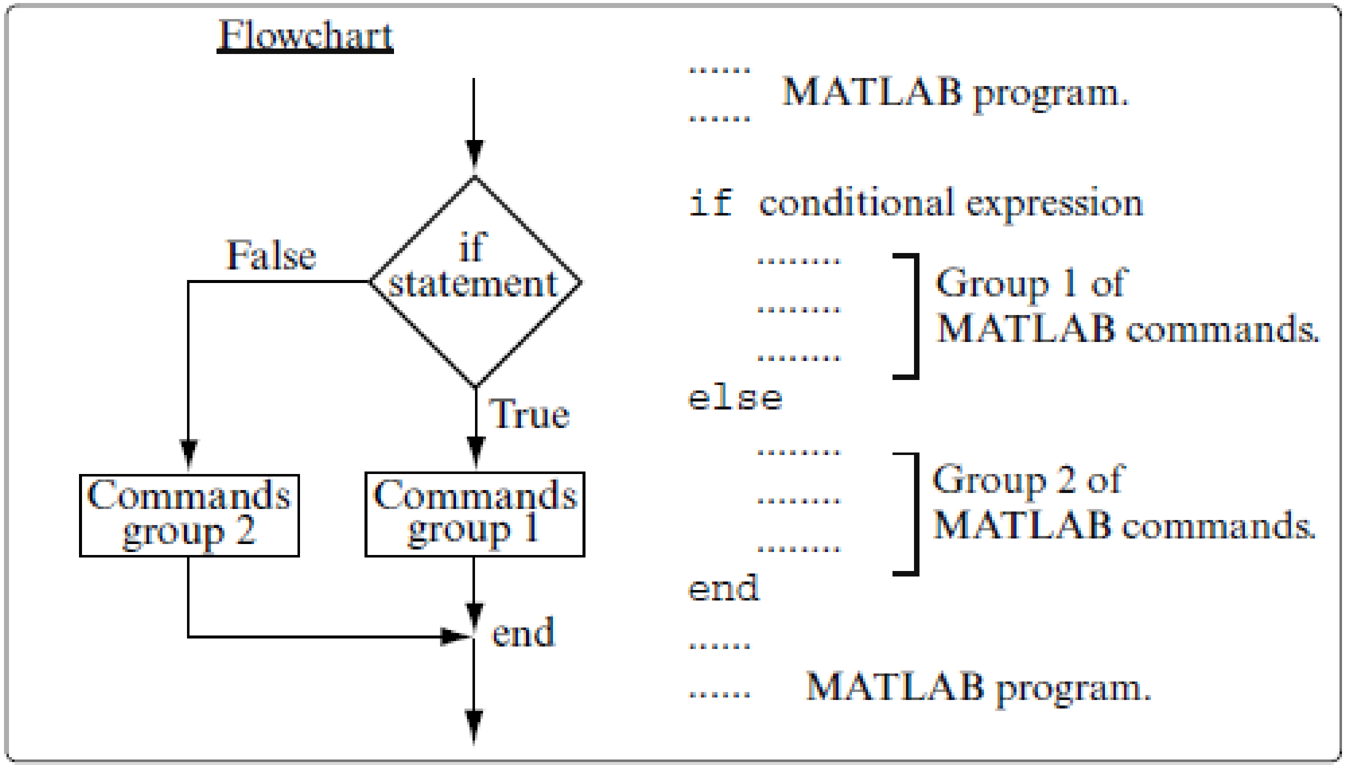
6.2.2 The `if-else-end` Structure

`if-else-end` structure lets you execute one section of code if a condition is true and a different section of code if it is false.

EXAMPLE - answering your phone

```
if  the caller is your best friend
        talk for a long time
else
        talk for a short time
end
```

Fig. 6-2 shows the code and the flowchart for the `if-else-end` structure



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

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

6.2.3 The `if-elseif-else-end` Structure

`if-elseif-else-end` structure lets you choose one of three (or more) sections of code to execute

EXAMPLE - answering your phone

```
if the caller is your best friend
  talk for a long time
elseif the caller is your study-mate
  talk until you get the answer to the hard problem
else
  say you'll call back later
end
```

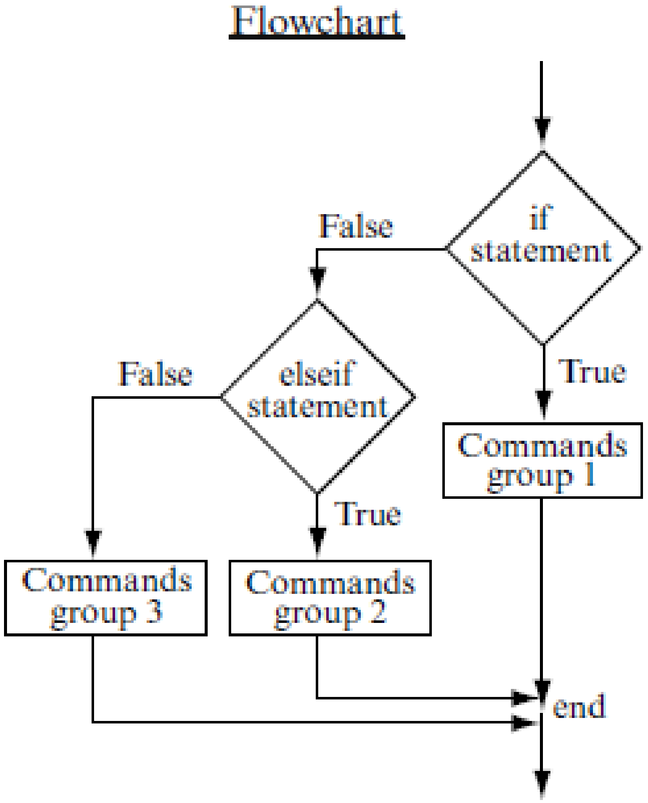
6.2.3 The `if-elseif-else-end` Structure

Can have as many `elseif` statements as you want

EXAMPLE

```
if  the caller is your best friend
   talk for a long time
elseif the caller is a potential date
   talk for a little bit and then set a time to meet
elseif the caller is your study-mate
   talk until you get the answer to the hard problem
elseif the caller is your mom
   say you're busy and can't talk
else
   have your room-mate say you'll call back later
end
```

Fig. 6-3 shows the code and the flowchart for the `if-elseif-else-end` structure



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

if conditional expression
       ........   ] Group 1 of
       ........   ] MATLAB commands.
       ........   ]
elseif conditional expression
       ........   ] Group 2 of
       ........   ] MATLAB commands.
       ........   ]
else
       ........   ] Group 3 of
       ........   ] MATLAB commands.
       ........   ]
end
......
......     MATLAB program.
```

6.2.3 The `if-elseif-else-end` Structure

Can omit `else` statement

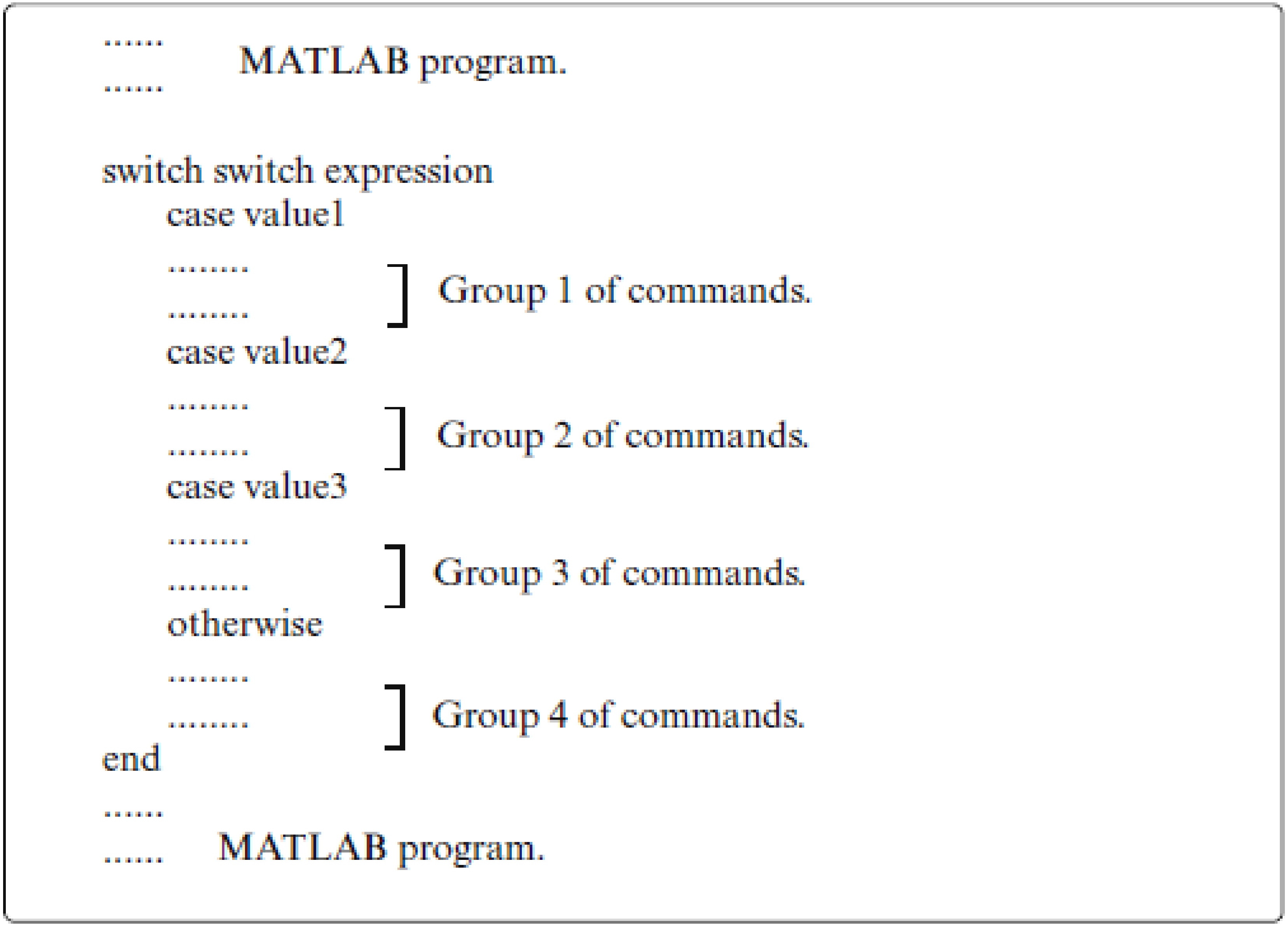
- In this case, if no match to `if`- or `elseif`-statements, no code in structure gets executed `if-elseif-else-end` structure gets hard to read if more than a few `elseif` statements. A clearer alternative is the `switch-case` structure

- `switch-case` slightly different because choose code to execute based on value of scalar or string, not just true/false

Concept is

```
switch name
case 'Bobby'
  talk for a long time
case 'Susan'
  talk for a little bit and then set a time to meet
case 'Hubert'
  talk until you get the answer to the hard problem
case 'Mom'
  say you're busy and can't talk
otherwise
  have your room-mate say you'll call back later
end
```

6.3 THE `switch-case` STATEMENT



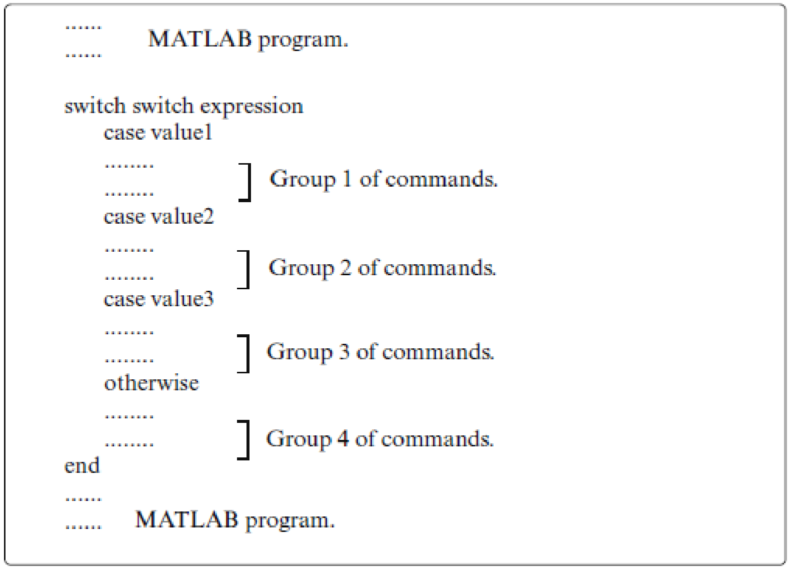
```
......
             MATLAB program.
......

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.
```

6.3 THE `switch-case` STATEMENT

`switch` evaluates switch-expression

- If value is equal to `value1`, executes all commands up to next `case`, `otherwise`, or `end` statement, i.e., Group 1 commands, then executes code after `end` statement
- If value is equal to `value2`, same as above but Group 2 commands only
- Etc.

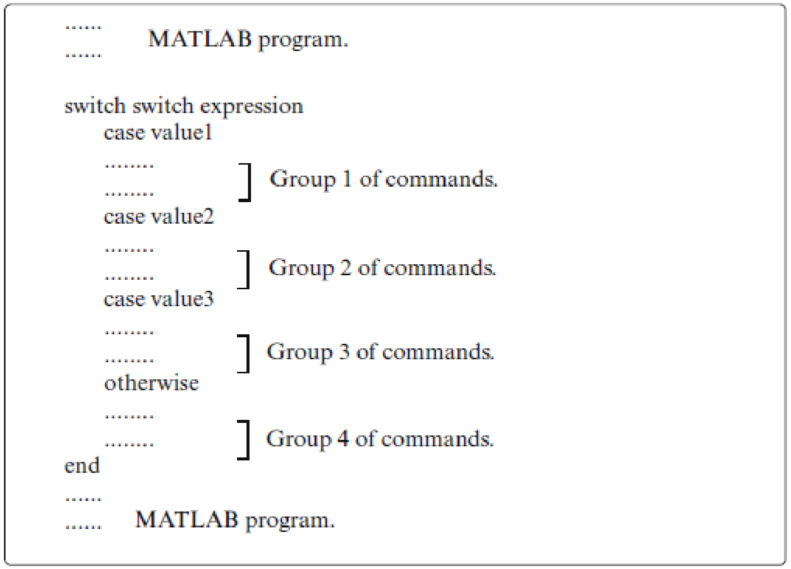


```
......      MATLAB program.
......

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.
```

6.3 THE `switch-case` STATEMENT

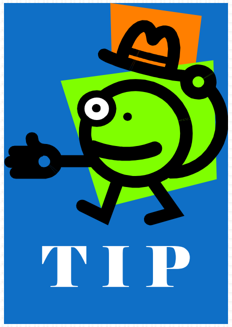
- If switch-expression not equal to any of values in `case` statement, commands after `otherwise` executed. If `otherwise` not present, no commands executed



```
......
......      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.
```

- If switch expression matches more than one `case` value, only first matching `case` executed



Comparisons of text strings are case-sensitive. If `case` values are text strings, make all values either lower case or upper case, then use `upper` or `lower` command to convert switch expression

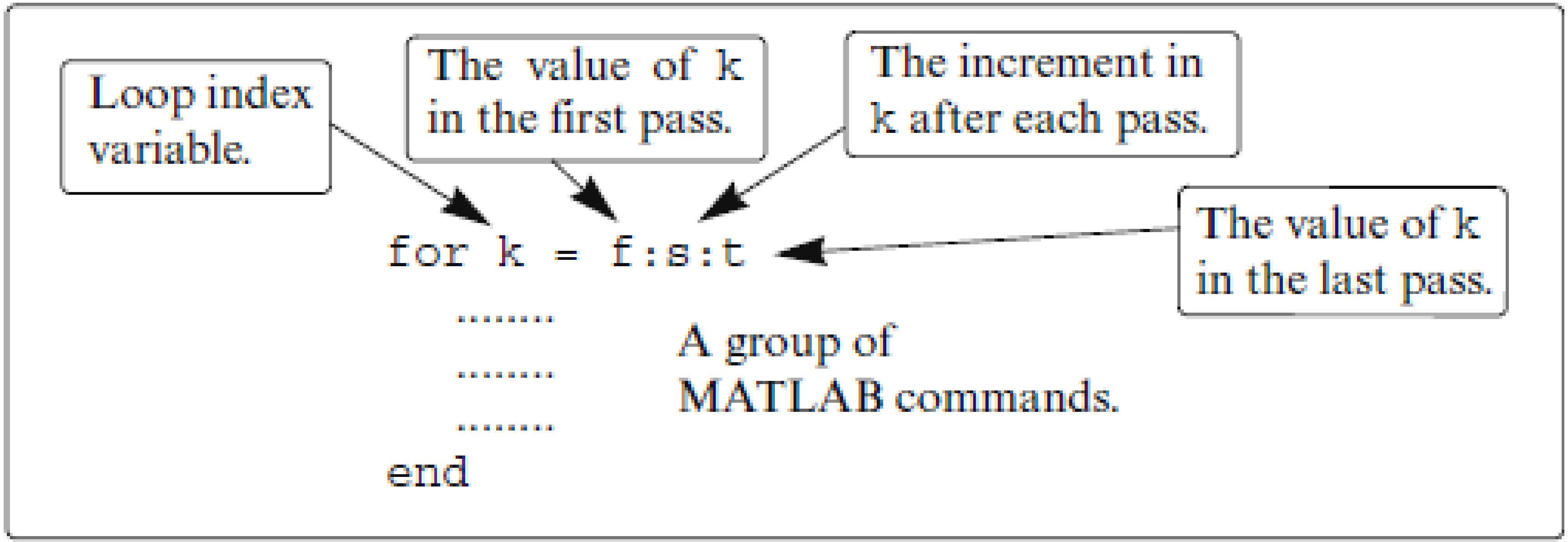
```
caller = lower( name );
switch caller
case 'bobby'
   some code
case 'susan'
   some code
case 'mom'
   some code
end
```

A *loop* is another method of flow control. A loop executes one set of commands repeatedly. MATLAB has two ways to control number of times loop executes commands

- Method 1 – loop executes commands a specified number of times
- Method 2 – loop executes commands as long as a specified expression is true

A `for-end` loop (often called a *for-loop*) executes set of commands a specified number of times. The set of commands is called the *body* of the loop

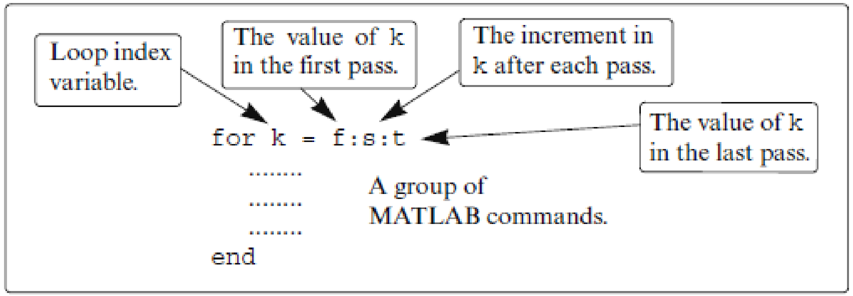
- The loop index variable can have any variable name (usually i, j, k, m, and n are used)
    - i and j should not be used when working with complex numbers. (ii and jj are good alternative names)



```
for k = f:s:t
    ........
    ........
    ........
end
```

Loop index variable.
The value of k in the first pass.
The increment in k after each pass.
The value of k in the last pass.
A group of MATLAB commands.

6.4.1 `for-end` Loops



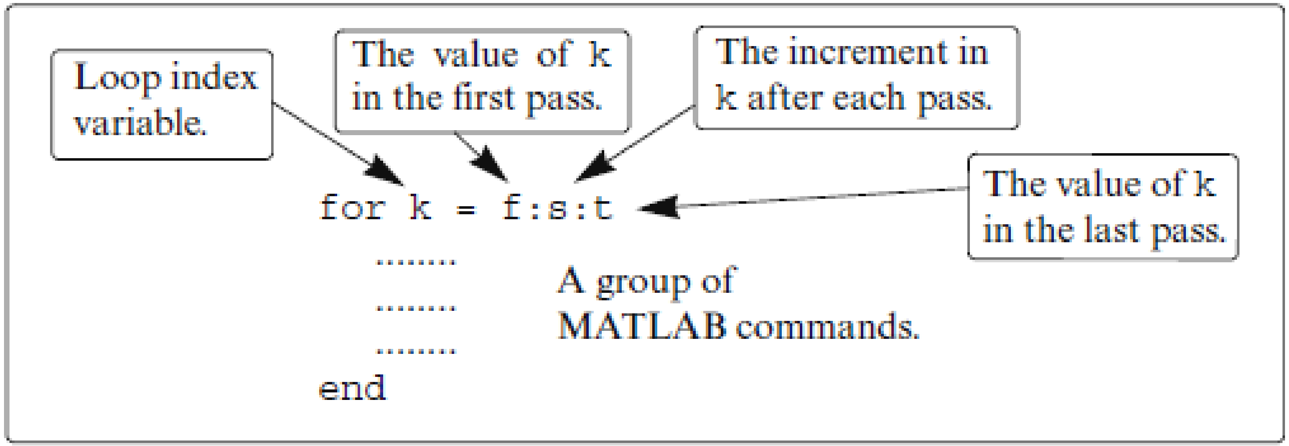
```
for k = f:s:t
    ........
    ........
    ........
end
```

Loop index variable.
The value of k in the first pass.
The increment in k after each pass.
The value of k in the last pass.
A group of MATLAB commands.

1. Loop sets `k` to `f`, and executes commands between `for` and the `end` commands, i.e., executes body of loop
2. Loop sets `k` to `f+s`, executes body
3. Process repeats itself until `k > t`
4. Program then continues with commands that follow `end` command

- `f` and `t` are usually integers
- `s` usually omitted. If so, loop uses increment of 1

- Increment `s` can be negative
  - For example, `k = 25:-5:10` produces four passes with `k = 25, 20, 15, 10`
- If `f = t`, loop executes once
- If `f > t` and `s > 0`, or if `f < t` and `s < 0`, loop not executed

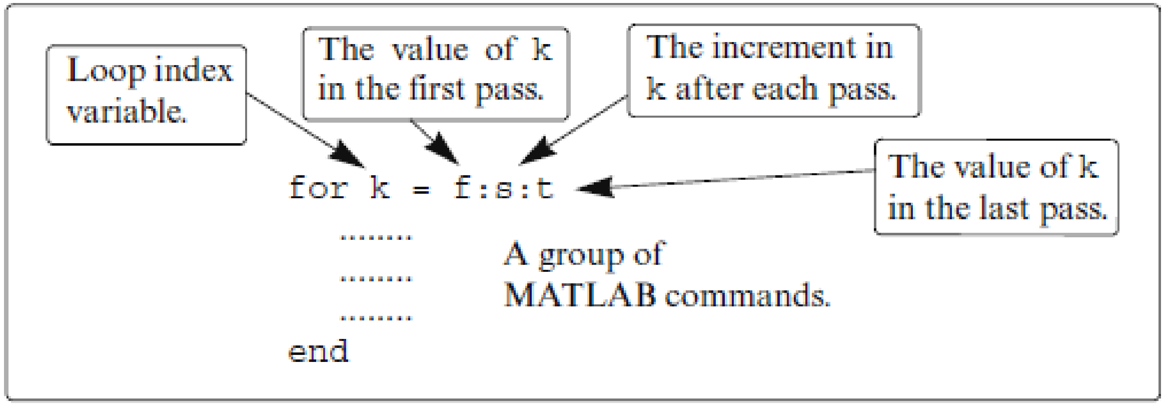


```
for k = f:s:t
    ........
    ........
    ........
end
```

Loop index variable.
The value of k in the first pass.
The increment in k after each pass.
The value of k in the last pass.
A group of MATLAB commands.

6.4.1 `for-end` Loops

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

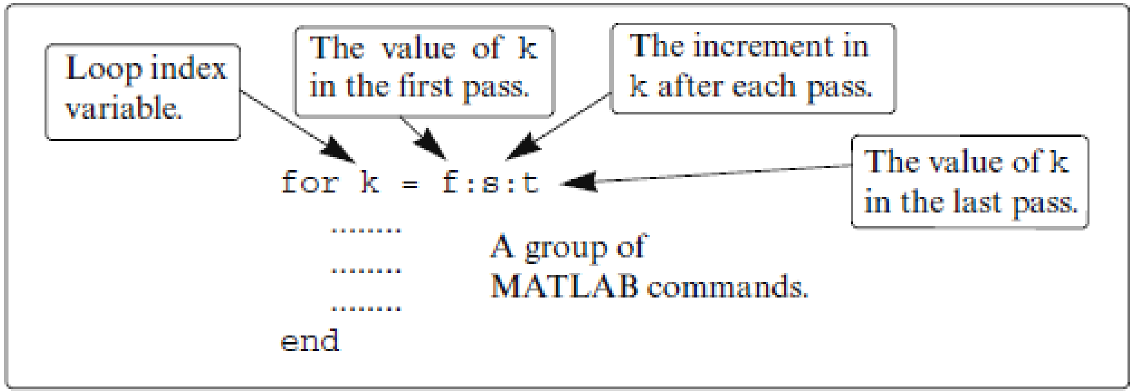


```
for k = f:s:t
    ........
    ........
    ........
end
```

Loop index variable.
The value of k in the first pass.
The increment in k after each pass.
The value of k in the last pass.
A group of MATLAB commands.

6.4.1 `for-end` Loops

- In the `for` command `k` can also be assigned specific value (typed in as a vector)
  - For example: `for k = [7 9 -1 3 3 5]`
- In general, loop body should not change value of `k`
- Each `for` command in a program <u>must</u> have an `end` command



Loop index variable.

The value of k in the first pass.

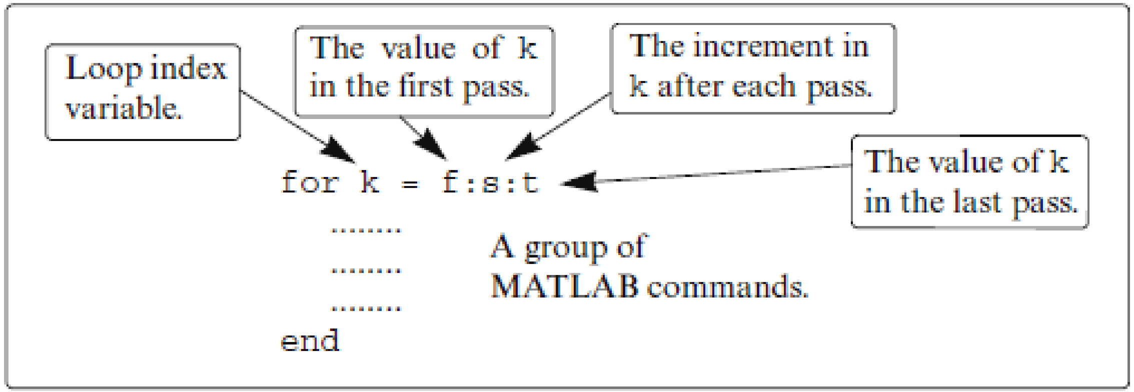
The increment in k after each pass.

The value of k in the last pass.

```
for k = f:s:t
    ........
    ........
    ........
end
```

A group of MATLAB commands.

- Value of loop index variable (k) not displayed automatically
    - Can display value in each pass (sometimes useful for debugging) by typing k as one of commands in loop
- When loop ends, loop index variable (k) has value last assigned to it



```
for k = f:s:t
    ........
    ........
    ........
end
```

Loop index variable.
The value of k in the first pass.
The increment in k after each pass.
The value of k in the last pass.
A group of MATLAB commands.

EXAMPLE

Script

```
for k=1:3:10
      k
      x = k^2
end
fprintf('After loop k = %d\n', k);
```

Output

```
k =    1
x =    1
k =    4
x =   16
k =    7
x =   49
k =   10
x =  100
After loop k = 10
```

Can often calculate something using either a for-loop or elementwise operations.

Elementwise operations are:

- Often faster
- Often easier to read
- More MATLAB-like

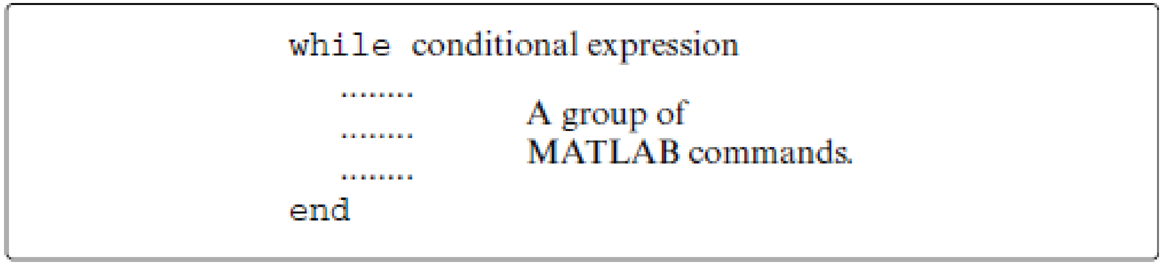
TIP

GENERAL ADVICE – use elementwise operations when you can, for-loops when you have to

`while-end` loop used when

- You don't know number of loop iterations
- You do have a condition that you can test and stop looping when it is false. For example,
  - Keep reading data from a file until you reach the end of the file
  - Keep adding terms to a sum until the difference of the last two terms is less than a certain amount

6.4.2 `while-end` Loops



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

1. Loop evaluates conditional-expression
2. If conditional-expression is true, executes code in body, then goes back to Step 1
3. If conditional-expression is false, skips code in body and goes to code after `end`-statement

The conditional expression of a `while-end` loop

- Has a variable in it
  - Body of loop must change value of variable
  - There must be some value of the variable that makes the conditional expression be false

EXAMPLE

This script

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

Makes this output

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

If the conditional expression never becomes false, the loop will keep executing… forever! The book calls this an *indefinite loop*, but more commonly referred to as an *infinite loop*. Your program will just keep running, and if there is no output from the loop (as if often the case), it will look like MATLAB has stopped responding

Common causes of indefinite loops:

- No variable in conditional expression

```
distance1 = 1;
distance2 = 10;
distance3 = 0;
while distance1 < distance2
   fprintf('Distance = 
   %d\n',distance3);
end
```

distance1 and distance2 never change

Common causes of indefinite loops:

- Variable in conditional expression never changes

```
minDistance = 42;
distanceIncrement = 0; ← Typo – should be 10
distance = 0;
while distance < minDistance
   distance = distance+distanceIncrement;
end
```

Common causes of indefinite loops:

- Wrong variable in conditional expression changed

```
minDistance = 42;
delta = 10;
distance = 0;
while distance < minDistance
   minDistance = minDistance + delta;
end
```

Typo – should be distance

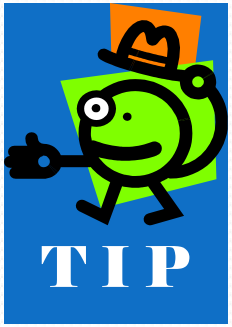
Common causes of indefinite loops:

- Conditional expression never becomes false

```
minDistance = 42;
x = 0;
y = 0;
while -sqrt( x^2+y^2 ) < minDistance
   x = x + 1;
   y = y + x;
end
```

Typo – shouldn't be any negative sign

6.4.2 `while-end` Loops



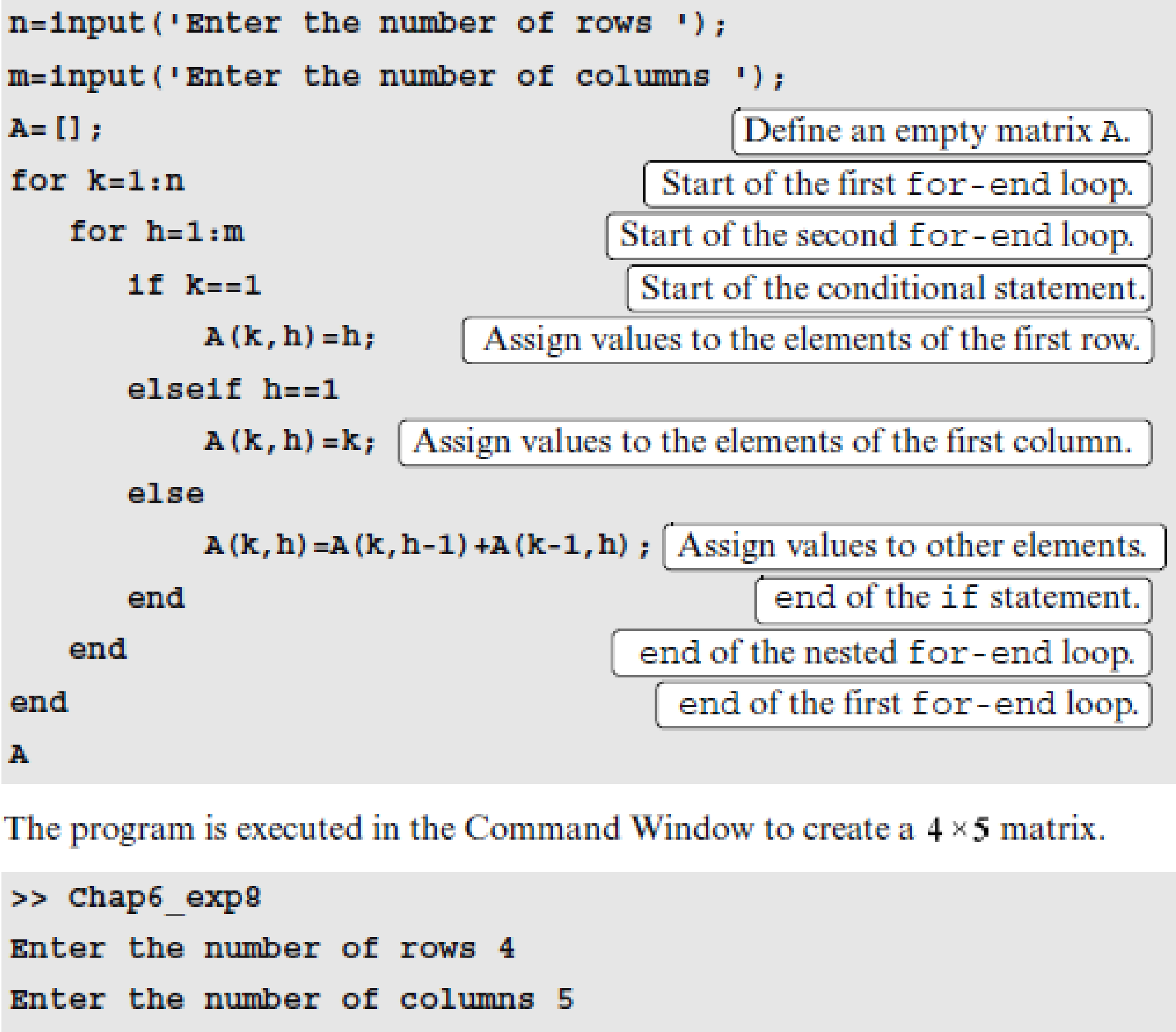
If your program gets caught in an indefinite loop,

- Put the cursor in the Command Window
- Press CTRL+C

If a loop or conditional statement is placed inside another loop or conditional statement, the former are said to be *nested* in the latter.

- Most common to hear of a *nested loop*, i.e., a loop within a loop
  - Often occur when working with two-dimensional problems
- Each loop and conditional statement <u>must</u> have an `end` statement

EXAMPLE



```
n=input('Enter the number of rows ');
m=input('Enter the number of columns ');
A=[];                                        Define an empty matrix A.
for k=1:n                                    Start of the first for-end loop.
   for h=1:m                                 Start of the second for-end loop.
      if k==1                                Start of the conditional statement.
          A(k,h)=h;                          Assign values to the elements of the first row.
      elseif h==1
          A(k,h)=k;                          Assign values to the elements of the first column.
      else
          A(k,h)=A(k,h-1)+A(k-1,h);          Assign values to other elements.
      end                                    end of the if statement.
   end                                       end of the nested for-end loop.
end                                          end of the first for-end loop.
A
```

The program is executed in the Command Window to create a 4 × 5 matrix.

```
>> Chap6_exp8
Enter the number of rows 4
Enter the number of columns 5
```

The `break` command:

- When inside a loop (for and while), `break` terminates execution of loop
  - MATLAB jumps from `break` to `end` command of loop, then continues with next command (does not go back to the `for` or `while` command of that loop).
  - `break` ends whole loop, not just last pass
- If `break` inside nested loop, only nested loop terminated (not any outer loops)

- `break` command in script or function file but not in a loop terminates execution of file
- `break` command usually used within a conditional statement.
  - In loops provides way to end looping if some condition is met

EXAMPLE

Script

Trick – "1" is always true so it makes loop iterate forever!

```
while( 1 )
  name = input( 'Type name or q to quit: ', 's' );
  if length( name ) == 1 && name(1) == 'q'
      break;
  else
      fprintf( 'Your name is %s\n', name );
  end
end
```

If user entered only one letter and it is a "q", jump out of loop

Only way to exit loop!

Otherwise print name

Output for inputs of "Greg", "quentin", "q"

```
Type name or q to quit: Greg
Your name is Greg
Type name or q to quit: quentin
Your name is quentin
Type name or q to quit: q
>>
```

The `continue` command:

Use `continue` inside a loop (for- and while-) to stop current iteration and start next iteration

- `continue` usually part of a conditional statement. When MATLAB reaches `continue` it does not execute remaining commands in loop but skips to the `end` command of loop and then starts a new iteration

EXAMPLE

```
for ii=1:100
   if rem( ii, 8 ) == 0
      count = 0;
      fprintf('ii=%d\n',ii);
      continue;
   end
   % code
   % more code
end
```

Every eight iteration reset `count` to zero, print the iteration number, and skip the remaining computations in the loop