Introduction to programming in MATLAB

Prof. G.J. Lanel

September 01, 2025

Outline

Outline

MATLAB windows

- When you start MATLAB, a special window called the MATLAB desktop appears.
- The desktop is a window that contains other windows. The major tools within or accessible from the desktop are:

MATLAB windows

- When you start MATLAB, a special window called the MATLAB desktop appears.
- The desktop is a window that contains other windows. The major tools within or accessible from the desktop are:

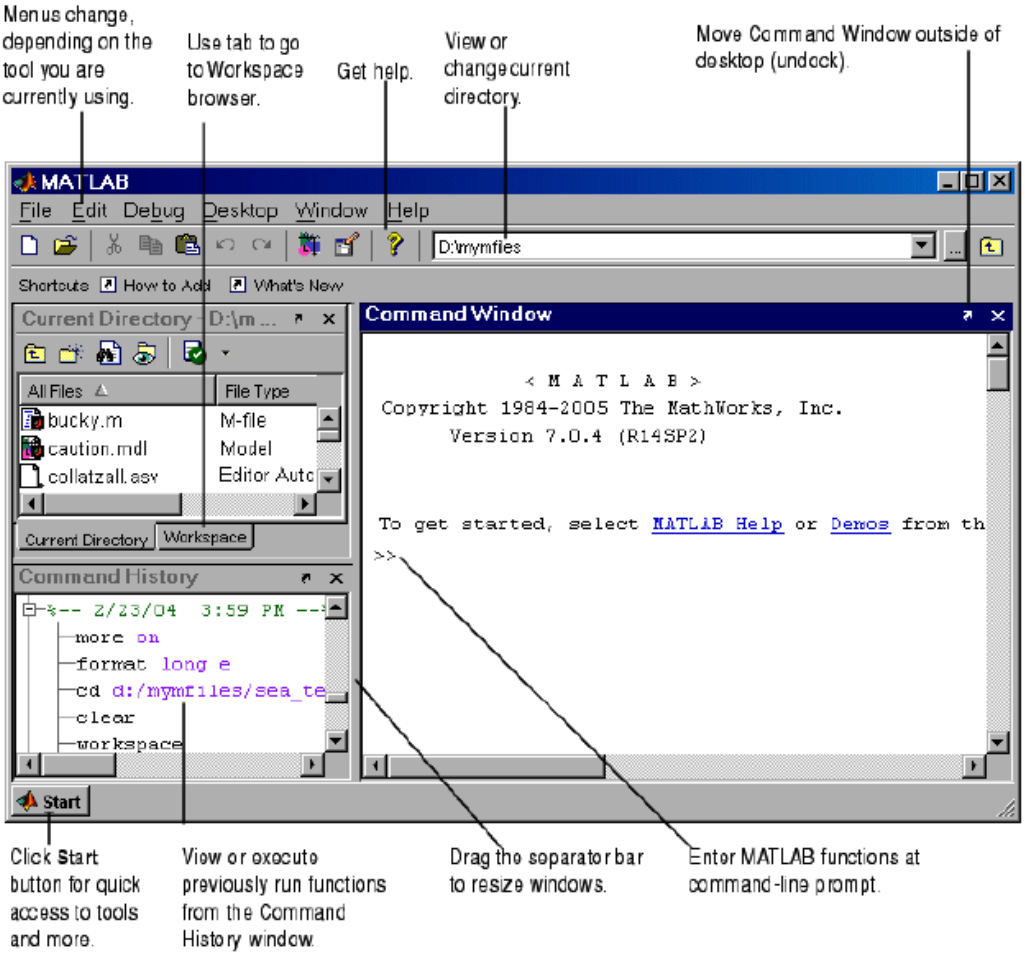
MATLAB windows

- When you start MATLAB, a special window called the MATLAB desktop appears.
- The desktop is a window that contains other windows. The major tools within or accessible from the desktop are:

Window	**Purpose**
Command window	Main window, enters variables, runs programs.
Editor window	Creates and debugs script and function files.
Help window	Provides help information.
Command History window	Logs command entered in the Command window
Workspace window	Provides information about the variables and that are used.
Current Directory window	Shows the files in the current directory.

Graphical interface of the MATLAB workspace

Graphical interface of the MATLAB workspace



Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of command the output of the command is not displayed.

Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of command the output of the command is not displayed.

Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of command the output of the command is not displayed.

Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of a command the output of the command is not displayed.

Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of a command the output of the command is not displayed.

Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of command the output of the command is not displayed.

Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of command the output of the command is not displayed.

Notes for working in the Command window

- To type a command the cursor must be placed next to the command prompt(»)
- Once a command is typed and the **Enter** key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Typed commands can be recalled to the command prompt with the up and down arrow keys(↑ & ↓).

The semicolon (;)

- Any output that the command generates is displayed in the Command window.
- If the semicolon is typed at the end of command the output of the command is not displayed.

Comments (%)

- When the symbol % is typed in the beginning of a line, the line is designated as a comment.

Order of precedence

Comments (%)

- When the symbol % is typed in the beginning of a line, the line is designated as a comment.

Order of precedence

Comments (%)

- When the symbol % is typed in the beginning of a line, the line is designated as a comment.

Order of precedence

Comments (%)

- When the symbol % is typed in the beginning of a line, the line is designated as a comment.

Order of precedence

Precedence	Mathematical operation
First	Parentheses. For nested parentheses, the innermost are executed first.
Second	Exponentiation
Third	Multiplication, division(equal parenthesis)
Fourth	Addition and subtraction

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing **help elfun** and **help specfun** calls up full lists of elementary and special functions respectively.

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing help elfun and help specfun calls up full lists of elementary and special functions respectively.

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing **help elfun** and **help specfun** calls up full lists of elementary and special functions respectively.

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing help elfun and help specfun calls up full lists of elementary and special functions respectively.

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing help elfun and help specfun calls up full lists of elementary and special functions respectively.

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing **help elfun** and **help specfun** calls up full lists of elementary and special functions respectively.

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing **help elfun** and **help specfun** calls up full lists of elementary and special functions respectively.

Display formats

- The user can control the format in which MATLAB displays output on the screen.
- The format can be changed with the **format** command.
- MATLAB has several other formats for displaying numbers. Details of these formats can be obtained by typing **help format** in the Command window.

Mathematical functions

- MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.
- Typing **help elfun** and **help specfun** calls up full lists of elementary and special functions respectively.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Defining scalar variables

- A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.
- Once a variable is assigned a numerical value, it can be used in mathematical expressions, in functions, and in any MATLAB statement and commands.
- A variable is actually a name of memory location.

Rules about variable names

Variable name:

- Can be up to 63 characters long, can contain letters, digits, and the underscore character.
- Must begin with letter.
- Avoid using the names of a built-in function for a variable. Once a function name is used to define a variable, the function can not be used.

Predefined variables

- A number of frequently used variable are already defined when MATLAB is started. Some of the predefined variables are **ans, pi, eps, inf, i, j, NaN**

Useful commands for managing variables

- The following are commands that can be used to eliminate variables or to obtain information about variables that have been created.

Predefined variables

- A number of frequently used variable are already defined when MATLAB is started. Some of the predefined variables are **ans, pi, eps, inf, i, j, NaN**

Useful commands for managing variables

- The following are commands that can be used to eliminate variables or to obtain information about variables that have been created.

Predefined variables

- A number of frequently used variable are already defined when MATLAB is started. Some of the predefined variables are **ans, pi, eps, inf, i, j, NaN**

Useful commands for managing variables

- The following are commands that can be used to eliminate variables or to obtain information about variables that have been created.

Predefined variables

- A number of frequently used variable are already defined when MATLAB is started. Some of the predefined variables are **ans, pi, eps, inf, i, j, NaN**

Useful commands for managing variables

- The following are commands that can be used to eliminate variables or to obtain information about variables that have been created.

Command	Outcome
clear	Removes all variables from the memory.
clear x y z	Removes only variables x,y, and z from the memory
who	Displays a list of the variables currently in the memory.
whos	Displays a list of the variables currently in the memory and their size together with information about their bytes and class.

The **fprintf** Function

- The general form of the fprintf function is: **fprintf (format, data)**
- **format** is a string that controls the way the data is to be printed, and **data** is one or more scalars or arrays to be printed.
- The format is a character string containing text to be printed plus special characters describing the format of the data.

Example:

```
temp = 78.25456789;
fprintf('The temperature is %f degrees.\n',temp)
```

The **fprintf** Function

- The general form of the fprintf function is: **fprintf (format, data)**
- **format** is a string that controls the way the data is to be printed, and **data** is one or more scalars or arrays to be printed.
- The format is a character string containing text to be printed plus special characters describing the format of the data.

The **fprintf** Function

- The general form of the fprintf function is: **fprintf (format, data)**
- **format** is a string that controls the way the data is to be printed, and **data** is one or more scalars or arrays to be printed.
- The format is a character string containing text to be printed plus special characters describing the format of the data.

The **fprintf** Function

- The general form of the fprintf function is: **fprintf (format, data)**
- **format** is a string that controls the way the data is to be printed, and **data** is one or more scalars or arrays to be printed.
- The format is a character string containing text to be printed plus special characters describing the format of the data.

Example:

```
temp = 78.234567989;
fprintf('The temperature is %f degrees. \n',temp)
```

The **fprintf** Function

- The general form of the fprintf function is: **fprintf (format, data)**
- **format** is a string that controls the way the data is to be printed, and **data** is one or more scalars or arrays to be printed.
- The format is a character string containing text to be printed plus special characters describing the format of the data.

Example:

```
temp = 78.234567989;
fprintf('The temperature is %f degrees. \n',temp)
```

The **fprintf** Function

- The general form of the fprintf function is: **fprintf (format, data)**
- **format** is a string that controls the way the data is to be printed, and **data** is one or more scalars or arrays to be printed.
- The format is a character string containing text to be printed plus special characters describing the format of the data.

Example:

```
temp = 78.234567989;
fprintf('The temperature is %f degrees. \n',temp)
```

Common Special Characters in `fprintf` Format Strings

Format String	Results
%d	Display value as an integer
%e	Display value in exponential format
%f	Display value in floating point format
%g	Display value in either floating point or exponential format, whichever is shorter
%c	Display a single character
%s	Display a string of characters
\n	Skip to a new line

Outline

There are three main programming constructs:

- **Sequence**
  The Sequence construct refers to writing a group of programming statements in a sequence.
- **Branch (Selection)**
  The Branch construct enables us to change the flow of control if a given condition is satisfied.
- **Repetition (Loop)**
  The Loop construct enables the program to run a statement (or a group of statements) a number of times.

There are three main programming constructs:

- **Sequence**
  The Sequence construct refers to writing a group of programming statements in a sequence.
- **Branch (Selection)**
  The Branch construct enables us to change the flow of control if a given condition is satisfied.
- **Repetition (Loop)**
  The Loop construct enables the program to run a statement (or a group of statements) a number of times.

There are three main programming constructs:

- **Sequence**
  The Sequence construct refers to writing a group of programming statements in a sequence.
- **Branch (Selection)**
  The Branch construct enables us to change the flow of control if a given condition is satisfied.
- **Repetition (Loop)**
  The Loop construct enables the program to run a statement (or a group of statements) a number of times.

There are three main programming constructs:

- **Sequence**
  The Sequence construct refers to writing a group of programming statements in a sequence.
- **Branch (Selection)**
  The Branch construct enables us to change the flow of control if a given condition is satisfied.
- **Repetition (Loop)**
  The Loop construct enables the program to run a statement (or a group of statements) a number of times.

MATLAB script files

- Matlab programming codes are saved in files with extension.m. This gives rise to the so-called Matlab M-files.
- An M-file may contain a Matlab script or a Matlab function.
- A script file is a sequence of MATLAB commands, also called a program.
- When a script file is executed it runs in the order that they are written just as they typed in the command window.
- When script file has a command that generates and output, the output is displayed in the command window.
- Using a script file is convenient because it can be edited and executed in many times.

MATLAB script files

- Matlab programming codes are saved in files with extension.m. This gives rise to the so-called Matlab M-files.
- An M-file may contain a Matlab script or a Matlab function.
- A script file is a sequence of MATLAB commands, also called a program.
- When a script file is executed it runs in the order that they are written just as they typed in the command window.
- When script file has a command that generates and output, the output is displayed in the command window.
- Using a script file is convenient because it can be edited and executed in many times.

MATLAB script files

- Matlab programming codes are saved in files with extension.m. This gives rise to the so-called Matlab M-files.
- An M-file may contain a Matlab script or a Matlab function.
- A script file is a sequence of MATLAB commands, also called a program.
- When a script file is executed it runs in the order that they are written just as they typed in the command window.
- When script file has a command that generates and output, the output is displayed in the command window.
- Using a script file is convenient because it can be edited and executed in many times.

MATLAB script files

- Matlab programming codes are saved in files with extension.m. This gives rise to the so-called Matlab M-files.
- An M-file may contain a Matlab script or a Matlab function.
- A script file is a sequence of MATLAB commands, also called a program.
- When a script file is executed it runs in the order that they are written just as they typed in the command window.
- When script file has a command that generates and output, the output is displayed in the command window.
- Using a script file is convenient because it can be edited and executed in many times.

MATLAB script files

- Matlab programming codes are saved in files with extension.m. This gives rise to the so-called Matlab M-files.
- An M-file may contain a Matlab script or a Matlab function.
- A script file is a sequence of MATLAB commands, also called a program.
- When a script file is executed it runs in the order that they are written just as they typed in the command window.
- When script file has a command that generates and output, the output is displayed in the command window.
- Using a script file is convenient because it can be edited and executed in many times.

MATLAB script files

- Matlab programming codes are saved in files with extension.m. This gives rise to the so-called Matlab M-files.
- An M-file may contain a Matlab script or a Matlab function.
- A script file is a sequence of MATLAB commands, also called a program.
- When a script file is executed it runs in the order that they are written just as they typed in the command window.
- When script file has a command that generates and output, the output is displayed in the command window.
- Using a script file is convenient because it can be edited and executed in many times.

Outline

If statement

- The if statement in Matlab allows us to change the flow of control in our computer programs based on a specified condition.
- The simplest way of using the if statement is as follows:

```
if <condition>
    <statement 1>
    <statement 2>
    ...
end
```

If statement

- The if statement in Matlab allows us to change the flow of control in our computer programs based on a specified condition.
- The simplest way of using the if statement is as follows:

```
if <condition>
    statement 1
    statement 2
     .
     .
     .
end
```

If statement

- The if statement in Matlab allows us to change the flow of control in our computer programs based on a specified condition.
- The simplest way of using the if statement is as follows:

```
if <condition>
    statement 1
    statement 2
     .
     .
     .
end
```

If statement

- The if statement in Matlab allows us to change the flow of control in our computer programs based on a specified condition.
- The simplest way of using the if statement is as follows:

```
if <condition>
    statement 1
    statement 2
    .
    .
    .
end
```

Second form of using the if statement provides a way to test for a condition and execute the appropriate statement (or set of statements) if a condition is true or false.

```
if <condition>
    statement 1
    statement 2
    .
    .
    .
    else
    statement 1
    statement 2
    .
    .
    .
end
```

Second form of using the if statement provides a way to test for a condition and execute the appropriate statement (or set of statements) if a condition is true or false.

```
if <condition>
    statement 1
    statement 2
    .
    .
    .
    else
    statement 1
    statement 2
    .
    .
    .
end
```

The most general way of using the if statement is outlined below.

```
if <condition>
    statements
    elseif <condition>
        statements
    elseif <condition>
        statements
    .
    .
    else
        statements
end
```

The most general way of using the if statement is outlined below.

```
if <condition>
    statements
    elseif <condition>
        statements
    elseif <condition>
        statements
    .
    .
    else
        statements
end
```

- When discussing the if statement it is natural to discuss the logical operators, AND, OR, and NOT. In addition, we can use the relational operators in the conditional expressions.

- When discussing the if statement it is natural to discuss the logical operators, AND, OR, and NOT. In addition, we can use the relational operators in the conditional expressions.

Logical Operator	Matlab Representation
AND	&&
OR	||
NOT	~

Logical Operators in Matlab

Relational Operator	Matlab Representation
$<$ $\leq$	< <=
$>$ $\geq$	> >=
$=$	==
$\neq$	~=

Relational Operators in Matlab

Repetition(Loops)

- One of the strongest attributes of a computer is its ability to do fast repetitive operations on a set of data.
- we use this feature through loops when we want to repeat certain parts of our program over and over again.
- In MATLAB there are two basic forms of loop constructs: **for** loops and **while** loops.
- The major difference between these two types of loops is in how the repetition is controlled.
- The code in a for loop is repeated a specified number of times, and the number of repetitions is known before the loops starts.
- The code in a while loop is repeated an indefinite number of times until some user-specified condition is satisfied.

Repetition(Loops)

- One of the strongest attributes of a computer is its ability to do fast repetitive operations on a set of data.
- we use this feature through loops when we want to repeat certain parts of our program over and over again.
- In MATLAB there are two basic forms of loop constructs: **for** loops and **while** loops.
- The major difference between these two types of loops is in how the repetition is controlled.
- The code in a for loop is repeated a specified number of times, and the number of repetitions is known before the loops starts.
- The code in a while loop is repeated an indefinite number of times until some user-specified condition is satisfied.

Repetition(Loops)

- One of the strongest attributes of a computer is its ability to do fast repetitive operations on a set of data.
- we use this feature through loops when we want to repeat certain parts of our program over and over again.
- In MATLAB there are two basic forms of loop constructs: **for** loops and **while** loops.
- The major difference between these two types of loops is in how the repetition is controlled.
- The code in a for loop is repeated a specified number of times, and the number of repetitions is known before the loops starts.
- The code in a while loop is repeated an indefinite number of times until some user-specified condition is satisfied.

Repetition(Loops)

- One of the strongest attributes of a computer is its ability to do fast repetitive operations on a set of data.
- we use this feature through loops when we want to repeat certain parts of our program over and over again.
- In MATLAB there are two basic forms of loop constructs: **for** loops and **while** loops.
- The major difference between these two types of loops is in how the repetition is controlled.
- The code in a for loop is repeated a specified number of times, and the number of repetitions is known before the loops starts.
- The code in a while loop is repeated an indefinite number of times until some user-specified condition is satisfied.

Repetition(Loops)

- One of the strongest attributes of a computer is its ability to do fast repetitive operations on a set of data.
- we use this feature through loops when we want to repeat certain parts of our program over and over again.
- In MATLAB there are two basic forms of loop constructs: **for** loops and **while** loops.
- The major difference between these two types of loops is in how the repetition is controlled.
- The code in a for loop is repeated a specified number of times, and the number of repetitions is known before the loops starts.
- The code in a while loop is repeated an indefinite number of times until some user-specified condition is satisfied.

Repetition(Loops)

- One of the strongest attributes of a computer is its ability to do fast repetitive operations on a set of data.
- we use this feature through loops when we want to repeat certain parts of our program over and over again.
- In MATLAB there are two basic forms of loop constructs: **for** loops and **while** loops.
- The major difference between these two types of loops is in how the repetition is controlled.
- The code in a for loop is repeated a specified number of times, and the number of repetitions is known before the loops starts.
- The code in a while loop is repeated an indefinite number of times until some user-specified condition is satisfied.

for Loop

- To execute a statement (or group of statements) a specified number of times we use the for loop.
- The basic usage of the for loop is as follows.

```
for index = expression
    statement group (body of the loop)
end
```

- The expression usually takes the form of a vector in shortcut notation **first:increment:last**.
- The index of the for loop must be a variable.

for Loop

- To execute a statement (or group of statements) a specified number of times we use the for loop.
- The basic usage of the for loop is as follows.

```
for index = expression
    statement group (body of the loop)
end
```

- The expression usually takes the form of a vector in shortcut notation **first:increment:last**.
- The index of the for loop must be a variable.

for Loop

- To execute a statement (or group of statements) a specified number of times we use the for loop.
- The basic usage of the for loop is as follows.

```
for index = expression
    statement group (body of the loop)
end
```

- The expression usually takes the form of a vector in shortcut notation **first:increment:last**.
- The index of the for loop must be a variable.

for Loop

- To execute a statement (or group of statements) a specified number of times we use the for loop.
- The basic usage of the for loop is as follows.

```
for index = expression
    statement group (body of the loop)
end
```

- The expression usually takes the form of a vector in shortcut notation **first:increment:last**.
- The index of the for loop must be a variable.

Example 1

```
for i = 1 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 2

```
for i = 1 : 2 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 3

```
for i = 10 : -1 : 1
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 1

```
for i = 1 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 2

```
for i = 1 : 2 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 3

```
for i = 10 : -1 : 1
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 1

```
for i = 1 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 2

```
for i = 1 : 2 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 3

```
for i = 10 : -1 : 1
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 1

```
for i = 1 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 2

```
for i = 1 : 2 : 10
    fprintf('%d ', i);
end
fprintf('\n');
```

Example 3

```
for i = 10 : -1 : 1
    fprintf('%d ', i);
end
fprintf('\n');
```

Example

Calculate the summation of 1 + 2 + ... + 100.

```
sum = 0;
    for i=1:100
    sum = sum + i;
end
fprintf(' The summation is %d \n ' ,sum);
```

```
» The summation is 5050
```

Example

Calculate the summation of 1 + 2 + ... + 100.

```
sum = 0;
      for i=1:100
      sum = sum + i;
end
fprintf(' The summation is %d \n ' ,sum);
```

```
» The summation is 5050
```

Example

Calculate the summation of 1 + 2 + ... + 100.

```
sum = 0;
      for i=1:100
      sum = sum + i;
end
fprintf(' The summation is %d \n ' ,sum);
```

» The summation is 5050

Example

Given a natural number n form an n$\times$ n Hilbert matrix whose (i, j)-component is defined as

H(i,j) = $\frac{1}{(i+j-1)}$, display the matrix.

```
n = input('Enter the number of terms := '); % change n to any value
for i = 1 : n
    for j = 1 : n
        H(i,j) = 1/(i+j-1);
    end
end
disp(H);
```

Example

Given a natural number n form an n$\times$ n Hilbert matrix whose (i, j)-component is defined as

$$H(i,j) = \frac{1}{(i+j-1)}$$, display the matrix.

```
n = input('Enter the number of terms := '); % change n to any value
for i = 1 : n
    for j = 1 : n
        H(i,j) = 1/(i+j-1);
    end
end
disp(H);
```

while Loop

- To execute a statement (or group of statements) a specified condition we use the while loop.
- The basic usage of the while loop is as follows.

```
while expression
    statement group
end
```

- The controlling expression produces a logical value.
- If the expression is always true (for example, we made an mistake in the expression), the loop becomes an infinite loop and we need to use the **Ctrl+C** key to abort it.

while Loop

- To execute a statement (or group of statements) a specified condition we use the while loop.
- The basic usage of the while loop is as follows.

```
while expression
    statement group
end
```

- The controlling expression produces a logical value.
- If the expression is always true (for example, we made an mistake in the expression), the loop becomes an infinite loop and we need to use the **Ctrl+C** key to abort it.

while Loop

- To execute a statement (or group of statements) a specified condition we use the while loop.
- The basic usage of the while loop is as follows.

```
while expression
    statement group
end
```

- The controlling expression produces a logical value.
- If the expression is always true (for example, we made an mistake in the expression), the loop becomes an infinite loop and we need to use the **Ctrl+C** key to abort it.

while Loop

- To execute a statement (or group of statements) a specified condition we use the while loop.
- The basic usage of the while loop is as follows.

```
while expression
    statement group
end
```

- The controlling expression produces a logical value.
- If the expression is always true (for example, we made an mistake in the expression), the loop becomes an infinite loop and we need to use the **Ctrl+C** key to abort it.

- If the expression is true, the statement group will be executed. The process will be repeated until the expression becomes false.
- If the expression is false, the program will execute the first statement after the end of while loop.

Example: Calculate the summation of $1+2+3+\cdots+n$, where $n(\geq 0)$ is given.

```
n = input('Input n :');
sum = 0;
current = 1;
while current <= n
    sum = sum + current;
    current = current + 1;
end
fprintf('The summation is %d\n', sum);
```

- If the expression is true, the statement group will be executed. The process will be repeated until the expression becomes false.
- If the expression is false, the program will execute the first statement after the end of while loop.

Example: Calculate the summation of 1 + 2 + ... + n. where n(>0) is given

```
n = input('Input n : ');
sum = 0;
current = 1;
while current <= n
    sum = sum + current;
    current = current + 1;
end
fprintf(' The summation is %d \n ' ,sum);
```

- If the expression is true, the statement group will be executed. The process will be repeated until the expression becomes false.
- If the expression is false, the program will execute the first statement after the end of while loop.

Example: Calculate the summation of 1 + 2 + ... + n. where n(>0) is given

```
n = input('Input n : ');
sum = 0;
current = 1;
while current <= n
    sum = sum + current;
    current = current + 1;
end
fprintf(' The summation is %d \n ' ,sum);
```

- If the expression is true, the statement group will be executed. The process will be repeated until the expression becomes false.
- If the expression is false, the program will execute the first statement after the end of while loop.

Example: Calculate the summation of 1 + 2 + ... + n. where n(>0) is given

```
n = input('Input n : ');
sum = 0;
current = 1;
while current <= n
    sum = sum + current;
    current = current + 1;
end
fprintf(' The summation is %d \n ' ,sum);
```

- If the expression is true, the statement group will be executed. The process will be repeated until the expression becomes false.
- If the expression is false, the program will execute the first statement after the end of while loop.

Example: Calculate the summation of 1 + 2 + ... + n. where n(>0) is given

```
n = input('Input n : ');
sum = 0;
current = 1;
while current <= n
      sum = sum + current;
      current = current + 1;
end
fprintf(' The summation is %d \n ' ,sum);
```

Outline

Arrays

- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element

Arrays

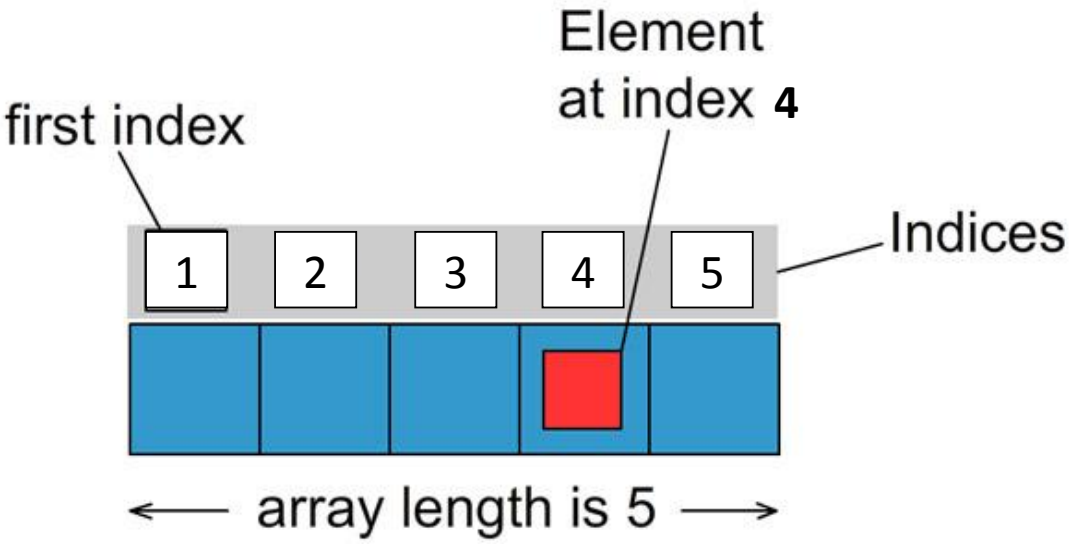
- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element

Arrays

- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element

Arrays

- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element



Vectors and Matrices

- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector**.
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns**.
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$

Vectors and Matrices

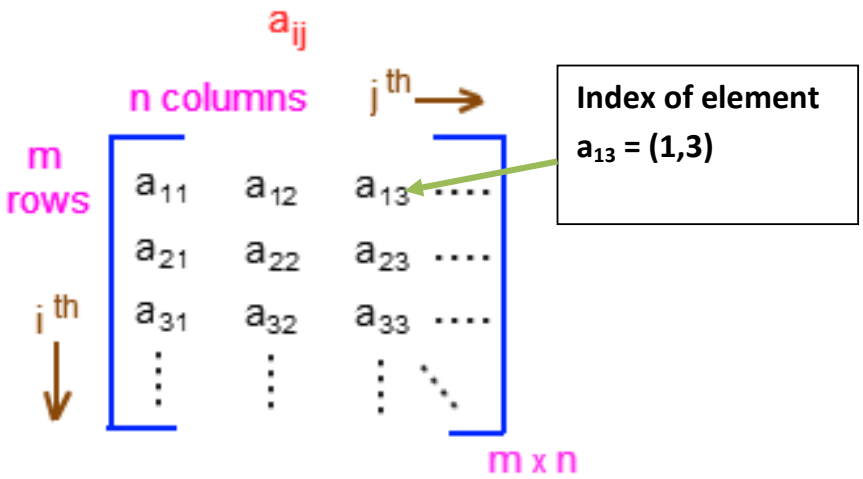
- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector**.
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns**.
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$

Vectors and Matrices

- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector**.
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns**.
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$

Vectors and Matrices

- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector**.
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns**.
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$



Basic Operations on Arrays

- **Defining an array** : vectors or matrices can be defined as follows
  - » A = [5 7 2 1] or A = [1,2,3,4] % Defining a row vector
  - » B = [3;6;2;9] % Defining a column vector
  - » C = [7 5; 8 9] % Defining 2 $\times$ 2 dimensional matrix
- **Access elements in arrays** :
  - » A(3) % 3 rd element of the vector A
  - » B(2,1) % index (2,1) element of the matrix B
  - » B(1,:) % All elements of the 1st row in matrix B
  - » B(:,2) % All elements of the 2nd column in matrix B
- Rows of a matrix can also be entered as vectors using the notation for creating vectors with **constant spacing**, or the **linspace** command.
  - » D = [1:2:11 ; 0:5:25 ; linspace(10,60,6) ; 67 32 4 58 9 18]

Basic Operations on Arrays

- **Defining an array** : vectors or matrices can be defined as follows
  - » A = [5 7 2 1] or A = [1,2,3,4] % Defining a row vector
  - » B = [3;6;2;9] % Defining a column vector
  - » C = [7 5; 8 9] % Defining 2 $\times$ 2 dimensional matrix
- **Access elements in arrays** :
  - » A(3) % 3 rd element of the vector A
  - » B(2,1) % index (2,1) element of the matrix B
  - » B(1,:) % All elements of the 1st row in matrix B
  - » B(:,2) % All elements of the 2nd column in matrix B
- Rows of a matrix can also be entered as vectors using the notation for creating vectors with **constant spacing**, or the **linspace** command.
  - » D = [1:2:11 ; 0:5:25 ; linspace(10,60,6) ; 67 32 4 58 9 18]

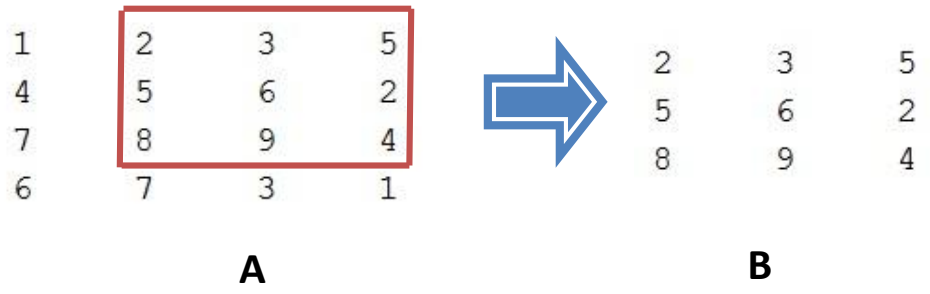
Basic Operations on Arrays

- **Defining an array** : vectors or matrices can be defined as follows
  - » A = [5 7 2 1] or A = [1,2,3,4] % Defining a row vector
  - » B = [3;6;2;9] % Defining a column vector
  - » C = [7 5; 8 9] % Defining 2 $\times$ 2 dimensional matrix
- **Access elements in arrays** :
  - » A(3) % 3 rd element of the vector A
  - » B(2,1) % index (2,1) element of the matrix B
  - » B(1,:) % All elements of the 1st row in matrix B
  - » B(:,2) % All elements of the 2nd column in matrix B
- Rows of a matrix can also be entered as vectors using the notation for creating vectors with **constant spacing**, or the **linspace** command.
  - » D = [1:2:11 ; 0:5:25 ; linspace(10,60,6) ; 67 32 4 58 9 18]

- **Deleting and inserting Elements** :
  - » B = [2 8 7 9 11 23 56 4 89 6];
  - » B(4) = 21; % insert 21 as 4th element
  - » B(3:6) = []; % remove elements from index 3 to 6
  - » B
- **Subset of an array** : subset of a vector or matrix can be obtained as follows
  - » A = [1 2 3 5; 4 5 6 2; 7 8 9 4;6 7 3 1]
  - » B = A(1:3,2:4) % subset of A

- **Deleting and inserting Elements** :
  - » B = [2 8 7 9 11 23 56 4 89 6];
  - » B(4) = 21; % insert 21 as 4th element
  - » B(3:6) = []; % remove elements from index 3 to 6
  - » B
- **Subset of an array** : subset of a vector or matrix can be obtained as follows
  - » A = [1 2 3 5; 4 5 6 2; 7 8 9 4;6 7 3 1]
  - » B = A(1:3,2:4) % subset of A

- **Deleting and inserting Elements** :
  - » B = [2 8 7 9 11 23 56 4 89 6];
  - » B(4) = 21; % insert 21 as 4th element
  - » B(3:6) = []; % remove elements from index 3 to 6
  - » B
- **Subset of an array** : subset of a vector or matrix can be obtained as follows
  - » A = [1 2 3 5; 4 5 6 2; 7 8 9 4;6 7 3 1]
  - » B = A(1:3,2:4) % subset of A



There are some useful elementary matrices in MATLAB

Elementary matrices

`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

Sometimes we have to perform arithmetic operations between the elements of two arrays of the same size in an element-by-element manner.

There are some useful elementary matrices in MATLAB

Elementary matrices

`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

Sometimes we have to perform arithmetic operations between the elements of two arrays of the same size in an element-by-element manner.

Summary of Array and Matrix operators

Character	Description
+ or -	Array **and** Matrix addition or subtraction of arrays
`.*`	Element-by-element multiplication of arrays
`./`	Element-by-element right division : a/b = `a(i,j)/b(i,j)`
`.\`	Element-by-element left division : a\b = `b(i,j)/a(i,j)`
`.^`	Element-by-element exponentiation
`*`	Matrix multiplication
`/`	Matrix right divide : `a/b = a*(b)`$^{-1}$
`\`	Matrix left divide (equation solve) : `a\b = (a)`$^{-1}$ `* b`
`^`	Matrix exponentiation

Outline

Functions

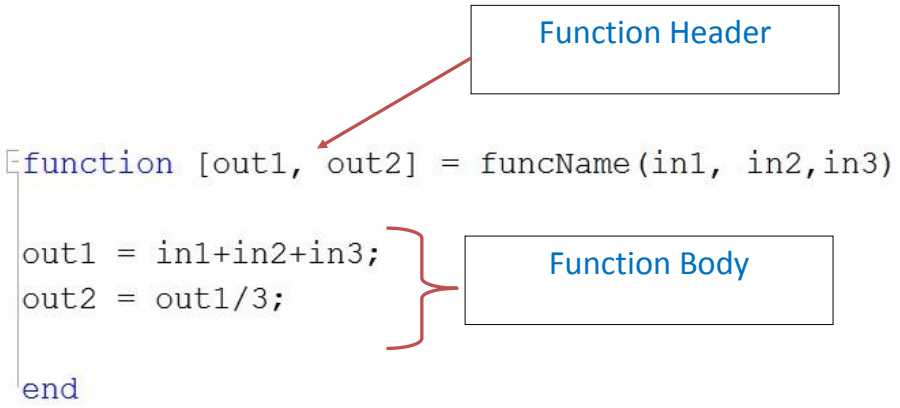
- Using functions to break down a large program to smaller and more manageable units is the heart of modular programming.
- In general, an m-file containing a Matlab function begins with the keyword **function** in the function header we specify the **name of the function** and the **input** and **output** parameters.

Functions

- Using functions to break down a large program to smaller and more manageable units is the heart of modular programming.
- In general, an m-file containing a Matlab function begins with the keyword **function** in the function header we specify the **name of the function** and the **input** and **output** parameters.

Functions

- Using functions to break down a large program to smaller and more manageable units is the heart of modular programming.
- In general, an m-file containing a Matlab function begins with the keyword **function** in the function header we specify the **name of the function** and the **input** and **output** parameters.



- Functions can have multiple inputs and multiple outputs

Example of input and output arguments

`function C=FtoC(F)`	One input argument and one output argument
`function area=TrapArea(a,b,h)`	Three inputs and one output
`function [h,d]=motion(v,angle)`	Two inputs and two outputs

- function file must be saved by the function name
- Similarly as in Maple function can be called by function name

- Functions can have multiple inputs and multiple outputs

Example of input and output arguments

`function C=FtoC(F)`	One input argument and one output argument
`function area=TrapArea(a,b,h)`	Three inputs and one output
`function [h,d]=motion(v,angle)`	Two inputs and two outputs

- function file must be saved by the function name
- Similarly as in Maple function can be called by function name

- Functions can have multiple inputs and multiple outputs

Example of input and output arguments

`function C=FtoC(F)`	One input argument and one output argument
`function area=TrapArea(a,b,h)`	Three inputs and one output
`function [h,d]=motion(v,angle)`	Two inputs and two outputs

- function file must be saved by the function name
- Similarly as in Maple function can be called by function name

Sub Functions and Main Function

- Defining a main function and sub functions is important in divide and conquer approach
- Main function and sub functions can be implemented on separate M-files. But they should be saved in the same directory
- You can also implement main function and sub functions in the same M-file as follows

Sub Functions and Main Function

- Defining a main function and sub functions is important in divide and conquer approach
- Main function and sub functions can be implemented on separate M-files. But they should be saved in the same directory
- You can also implement main function and sub functions in the same M-file as follows

Sub Functions and Main Function

- Defining a main function and sub functions is important in divide and conquer approach
- Main function and sub functions can be implemented on separate M-files. But they should be saved in the same directory
- You can also implement main function and sub functions in the same M-file as follows

```
function [sm,avg] = addavg(x,y)   % Main Function

sm = addition(x,y);
avg = aver(x,y);
end

function a = aver(x,y)            % Sub Function 01

a = addition(x,y)/2;
end

function s = addition(x,y)        % Sub Function 02

s = x+y;
end
```

Local and Global variables

- The variables defined in a function are recognized only inside the function file.
- It is possible, however , to make a variable to be recognized in different function files. In other words to make the variables are **global**.
- Then they all share a single copy of that variable. Any change of value to that variable, in any function, is visible to all other functions

Local and Global variables

- The variables defined in a function are recognized only inside the function file.
- It is possible, however , to make a variable to be recognized in different function files. In other words to make the variables are **global**.
- Then they all share a single copy of that variable. Any change of value to that variable, in any function, is visible to all other functions

Local and Global variables

- The variables defined in a function are recognized only inside the function file.
- It is possible, however , to make a variable to be recognized in different function files. In other words to make the variables are **global**.
- Then they all share a single copy of that variable. Any change of value to that variable, in any function, is visible to all other functions

Inline Function

- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

  Examples

  >> FA = inline('exp(x^2)/sqrt(x^2 + 5)')

  >> FA

  >> FA(2)

  >> f = inline('exp(x^2)/sqrt(x^2+y^2)','x','y');

  >> f

  >> f(2,3)

Inline Function

- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

Examples

```
» FA = inline('exp(x^2)/sqrt(x^2 + 5)');
» FA
» FA(2)
» f = inline('exp(x^2)/sqrt(x^2 + y^2)','x','y');
» f
» f(2,3)
```

Inline Function

- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

Examples

```
» FA = inline('exp(x^2)/sqrt(x^2 + 5)');
» FA
» FA(2)
» f = inline('exp(x^2)/sqrt(x^2 + y^2)','x','y');
» f
» f(2,3)
```

Inline Function

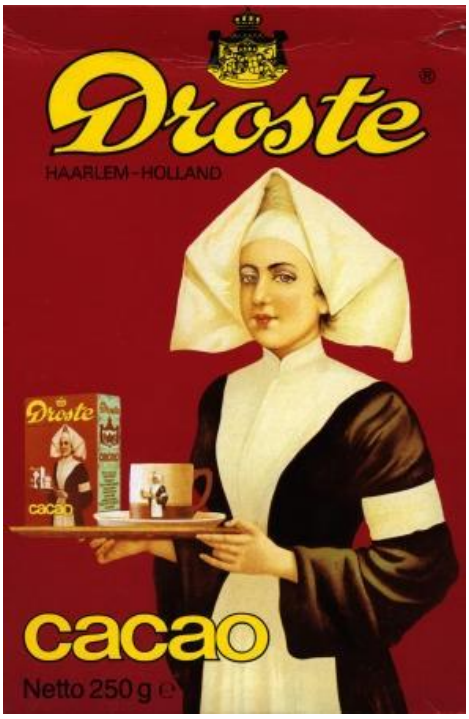
- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

Examples

» FA = inline('$exp(x^2)/sqrt(x^2+5)$');
» FA
» FA(2)
» f = inline('$exp(x^2)/sqrt(x^2+y^2)$','x','y');
» f
» f(2,3)

Recursion



Recursion is the process of repeating items in a self-similar way. The most common application of recursion is in mathematics and computer science, in which it refers to a method of defining functions in which the function being defined is applied within its own definition

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Example

Develop MATLAB function to calculate the sum of the first *n* integers using recursive formula

```
function [outsum] = sumrec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outsum = 1;
else
    outsum = sumrec(n-1) + n; % recursive formula
end
```

Example

Develop MATLAB function to calculate the sum of the first *n* integers using recursive formula

```
function [outsum] = sumrec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outsum = 1;
else
    outsum = sumrec(n-1) + n; % recursive formula
end
```

Example

Generating Fibonacci numbers : 0 1 1 2 3 5 8 13 21 ...
using recursive formula F(n) = F(n -1) + F(n - 2) ; F(0) = 0 and F(1) = 1

```
function [outfn] = fiborec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outfn = 0;
elseif n==2
    outfn = [0 1];
else
    fnm1 = fiborec(n-1);
    outfn = fnm1(n-1) + fnm1(n-2);
    outfn = [fnm1 outfn];
end
```

Example

Generating Fibonacci numbers : 0 1 1 2 3 5 8 13 21 ...
using recursive formula F(n) = F(n -1) + F(n - 2) ; F(0) = 0 and F(1) = 1

```
function [outfn] = fiborec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outfn = 0;
elseif n==2
    outfn = [0 1];
else
    fnm1 = fiborec(n-1);
    outfn = fnm1(n-1) + fnm1(n-2);
    outfn = [fnm1 outfn];
end
```

- Every recursive function must have a **terminating condition**. If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

- Every recursive function must have a **terminating condition**. If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

- Every recursive function must have a **terminating condition**. If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

- Every recursive function must have a **terminating condition**. If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

End!