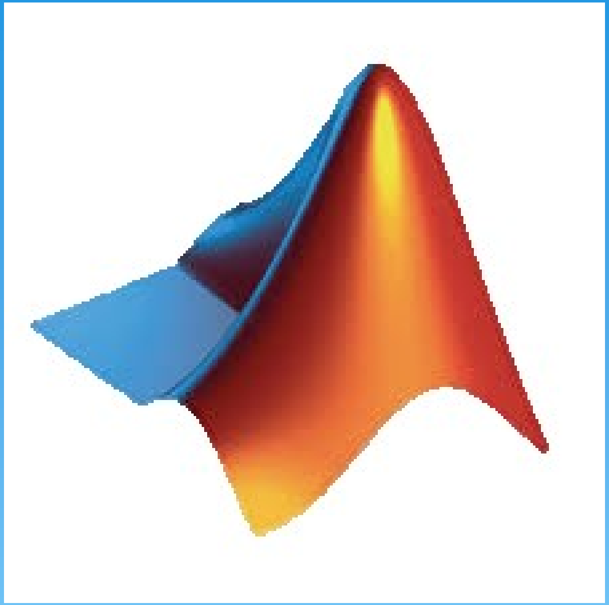


Introduction to MATLAB on Communication

Tutorial②

Dr. Victor B. Lawrence
& Ghalib Alshammri
galshamm@stevens.edu

CPE654: Design and Analysis of Network Systems
2017 Fall, Thursday 06:15 – 08:45 PM

Purpose and Objectives

- Learn about Communication Channel.
- Discover MATLAB environment.
- Learn about MATLAB features.
- Discover Communication toolbox at MATLAB.
- Develop a beginner level MATLAB application.

Outline

- Fundamental MATLAB
  - Control Flow
    - Decision Making
    - Loop Types
  - Creating a new function and calling
  - Vector, Matrix and Array
  - Graphics
    - Bar
    - Graphics in 2D and 3D
- Summary.

MATLAB

Control Flow

Decision Making

Decision Making Requirements:

* Specify one or more conditions to be evaluated.
* If the condition is determined to be true, a statement or statements will be executed or other statements to be executed if the condition is determined to be false.

Decision Making

Decision Making Statements:

Statements	Description
if …. end statement	It consists of a Boolean expression followed by one or more statements
if …. else … end statement	An if statement can be followed by an optional else statement, which executes when the Boolean expression is false
if … elseif … elseif … else … end statement	An if statement can be followed by one or more optional elseif … and an else statement, which is very useful to test various conditions

Decision Making

Decision Making Statements:

Statements	Description
nested if statements	Using one if or elseif statement inside another if or elseif statements
switch statement	A switch statement allows a variable to be tested for equality against a list of values
nested switch statement	Using one switch statement inside another switch statements

if … end Statement

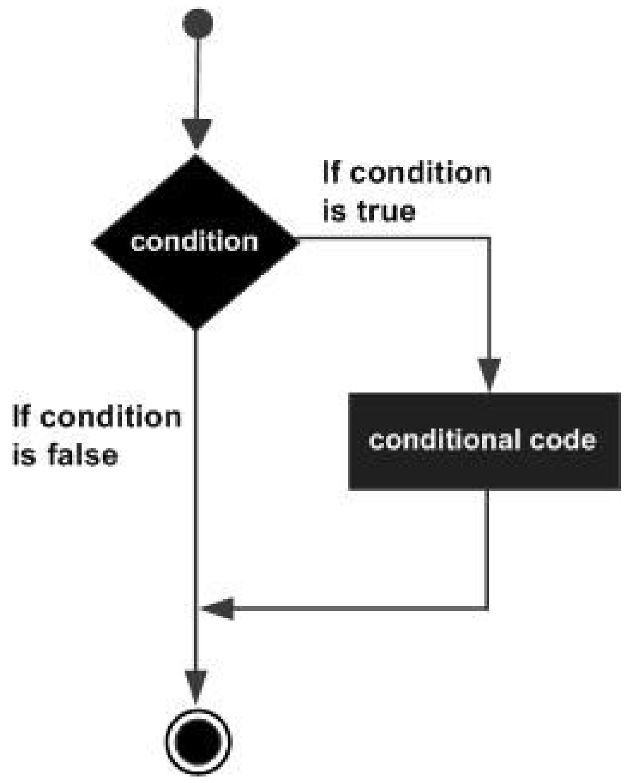
Syntax:

```
if (Boolean Expression)
        statement(s);
end
```

Example:

```
x = 25;
y = 10;

if (x >= y)
        disp(x);
end
```

```
25
```



if … else … end Statement

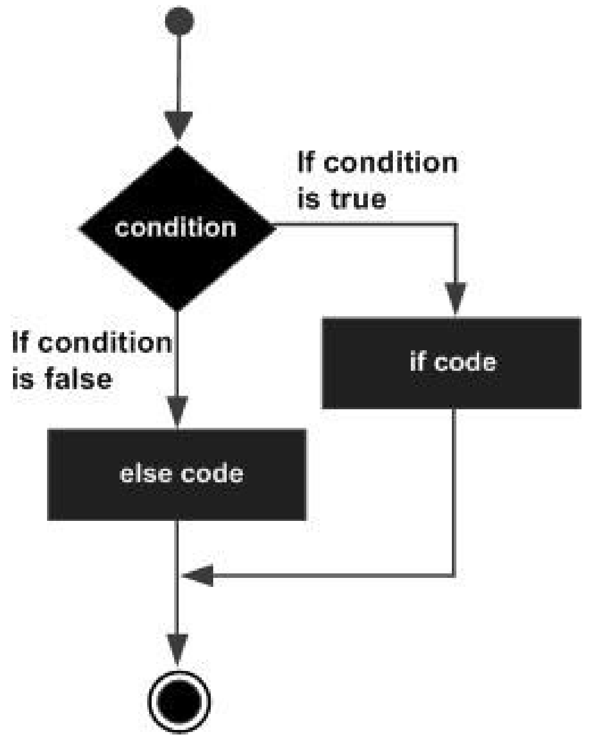
Syntax:

```
if (Boolean Expression)
        statement(s) if True;
else
        statement(s) if False;
end
```

Example:

```
x = 25;
y = 10;

if (x == y)
        disp(x);
else
        disp(y);
end
```

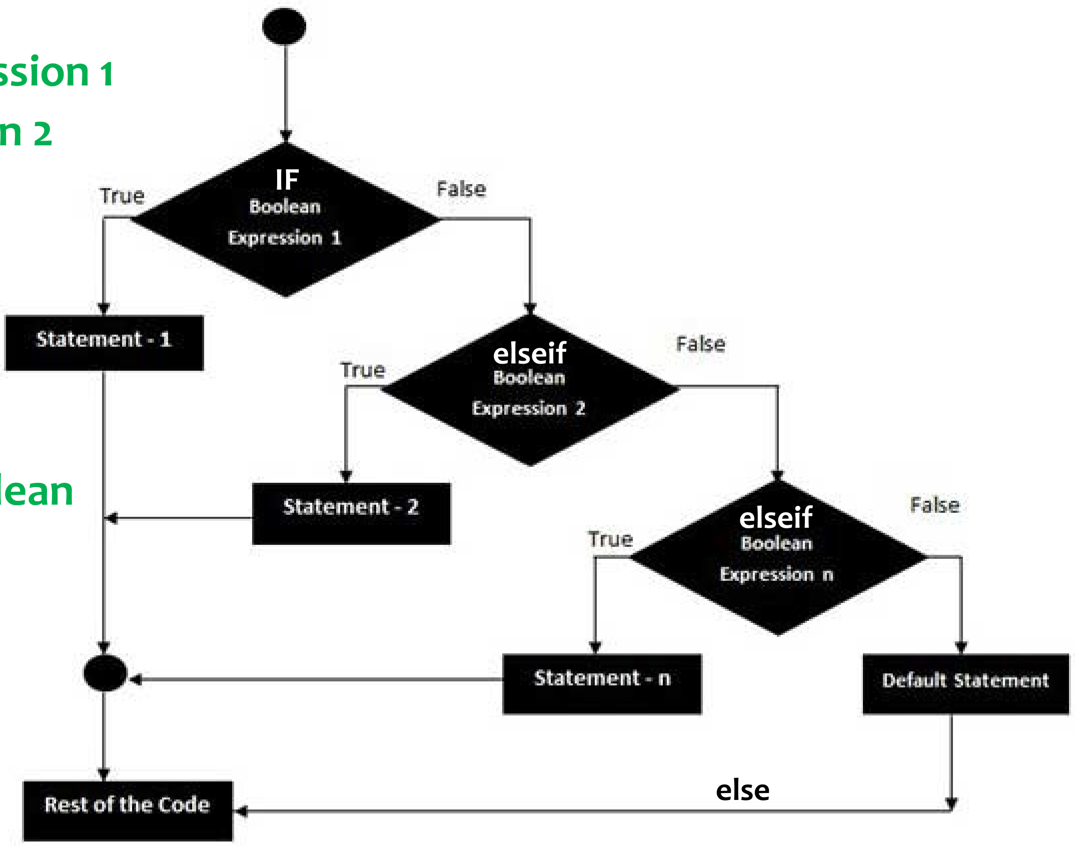
```
10
```



if … elseif … else … end Statement

Syntax:

```
if (Boolean Expression 1)
        statement(s) if Boolean Expression 1 is True;
elseif (Boolean Expression 2)
        statement(s) if Boolean Expression 1
        is False and Boolean Expression 2
        is True;
elseif ……….
elseif ……….
elseif (Boolean Expression n)
        statement(s) if all Boolean
        Expressions are False and Boolean
        Expression n is True;
else
        statement(s) if all Boolean
        Expressions are False;

end
```

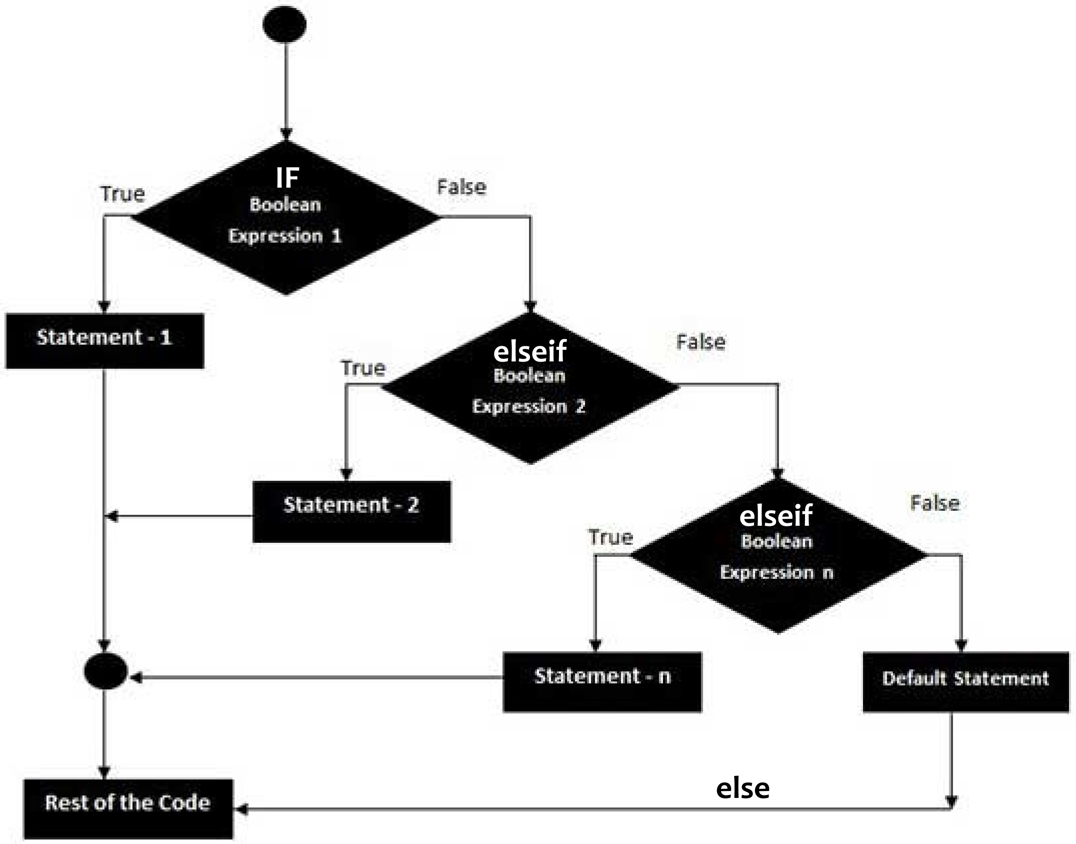


if … elseif … else … end Statement

Example:

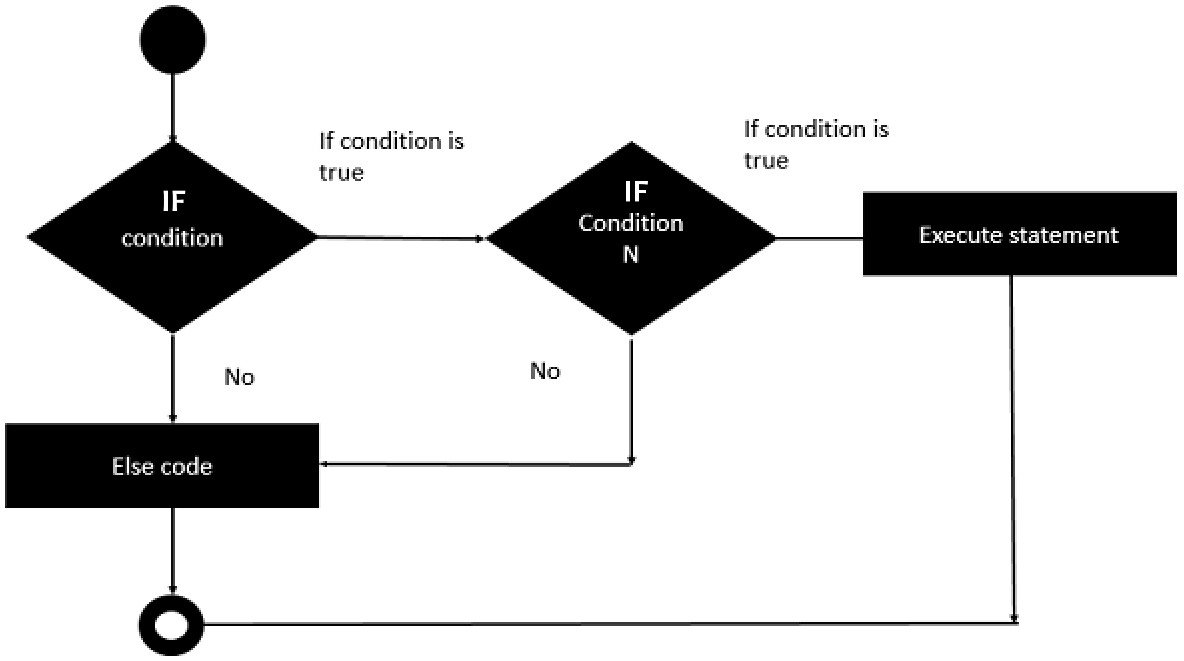
```
x  = 25;
y = 10;
if (x > y)
        disp('x is greater than y');
elseif (y > x)
        disp('y is greater than x');
else
        disp('x and y are equal');

end
```

```
x is greater than y
```



Nested if Statements

- Nested if statement is the most heavily used function.
- **What's nesting means?**
  "Nested simply means to combine formulas, one inside the other"



Nested if Statements

Syntax:

```
if (Boolean Expression 1)
        if (Boolean Expression 2)
                statement(s) or nested if statement(s)
                if exp. 1 and exp2 are True;
        else
                statement(s) or nested if statement(s)
                if exp. 1 is True and exp. 2 is False;
        end
else
        if (Boolean Expression 3)
                statement(s) or nested if statement(s)
                if exp. 1 is False and exp. 3 is True;
        else
                statement(s) or nested if statement(s)
                if exp. 1 is False and exp. 2 is False;
        end
end
```

Nested if Statements

Example:

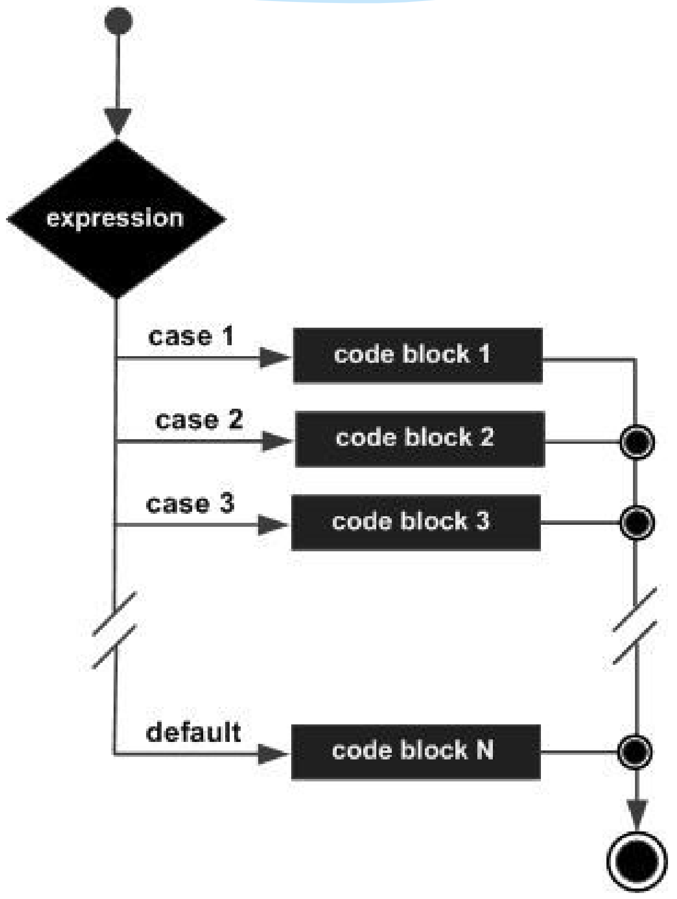
```
x = 50;
if ( x < 10)
            disp('x is less than 10');
else
            if (x < 7)
                        disp('x is less than 7');
            else
                        disp('x is greater than 10');
            end
end
```

```
x is greater than 10
```

Switch Statement

Syntax:

```
switch switch_expression or value
  case case_expression 1 or value 1
    statement(s) if case_expression 1 is True;
  case case_expression 2 or value 2
    statement(s) if case_expression 2 is True;
   ...
  case case_expression n or value n
    statement(s) if case_expression n is True;
  otherwise
    statement(s) if all cases are False;
end
```

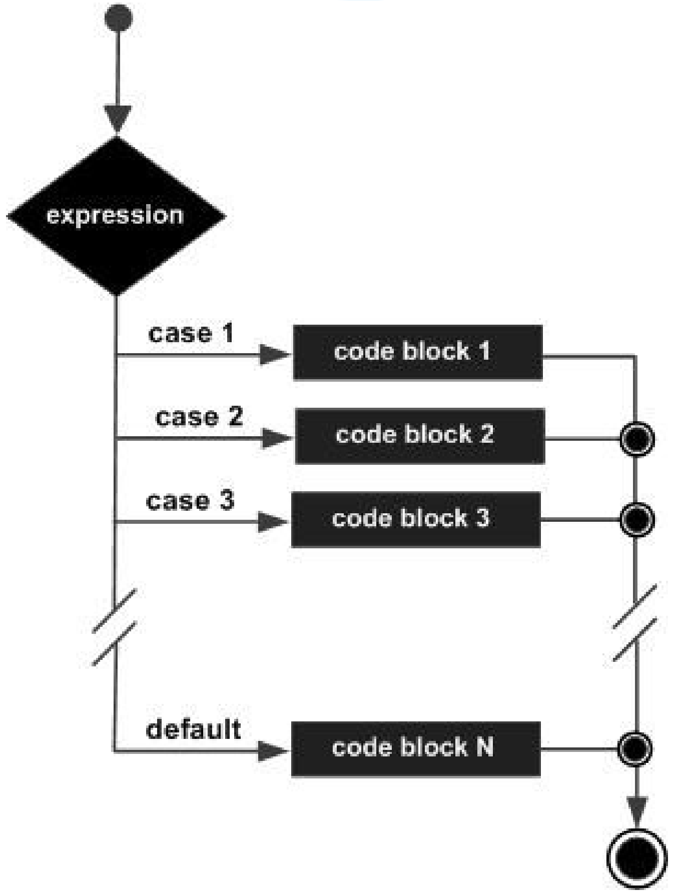


Switch Statement

Example:

```
n = input('Enter a number: ');

switch n
   case -1
      disp('negative one');
   case 0
      disp('zero');
   case 1
      disp('positive one');
   otherwise
      disp('other value');
end
```

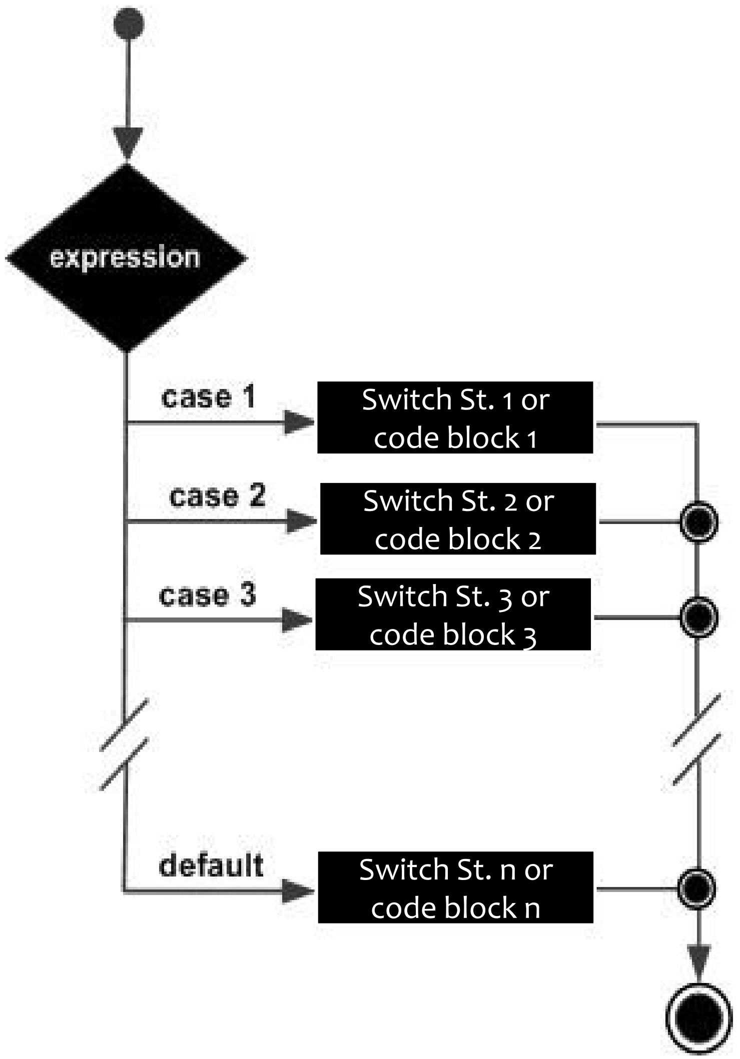
```
-1
 negative one
```



Nested Switch Statement

Syntax:

```
switch switch_expression or value
  case case_expression 1 or value 1
    statement(s) or switch statement(s)
    if case_expression 1 is True;
  case case_expression 2 or value 2
    statement(s) or switch statement(s)
    if case_expression 2 is True;
  ...
  case case_expression n or value n
    statement(s) or switch statement(s)
    if case_expression n is True;
  otherwise
    statement(s) or switch statement(s)
    if all cases are False;
end
```

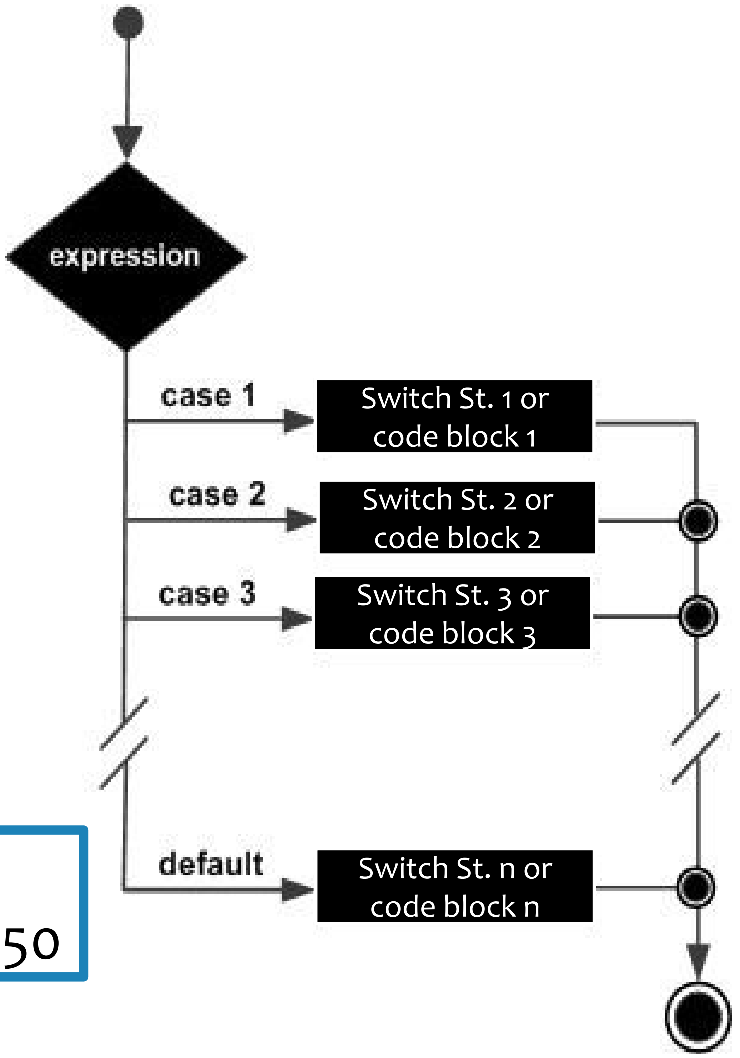

expression
case 1
Switch St. 1 or code block 1
case 2
Switch St. 2 or code block 2
case 3
Switch St. 3 or code block 3
default
Switch St. n or code block n

Nested Switch Statement

Example:

```
n = input('Enter a number: ');

switch True
   case n <= -1
         switch True
              case n >= -50
                     disp(‘negative and greater than -50’);
              otherwise
                     disp(‘negative and less than -50’);
         end
   case n == 0
      disp('zero');
   case n >= 1
         switch True
              case n <= 50
                     disp(‘positive and less than 50’);
              otherwise
                     disp(‘positive and greater than 50’);
         end
   otherwise
      disp('other value');
end
```

```
-34
negative and greater than -50
```


expression
case 1
Switch St. 1 or code block 1
case 2
Switch St. 2 or code block 2
case 3
Switch St. 3 or code block 3
default
Switch St. n or code block n

Loop Types

Loop Type Requirements:

- Executing a block of code several number of times.
- A loop statement allows programmers to execute a statement or group of statements multiple times when a loop condition is True.
- A loop statement need a counter to ensure the continuity of a loop statement.

Loop Types

Loop Statements:

Statements	Description
while loop	Repeats a statement or group of statements while a given condition is True. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the variable.
nested loops	Using one or more loops inside any other loop.

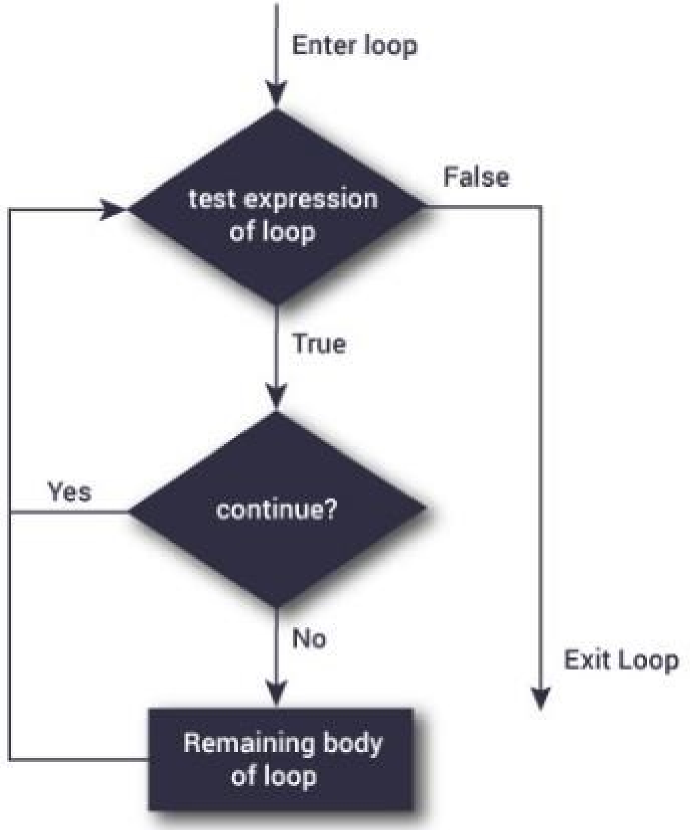
Loop Control Statements

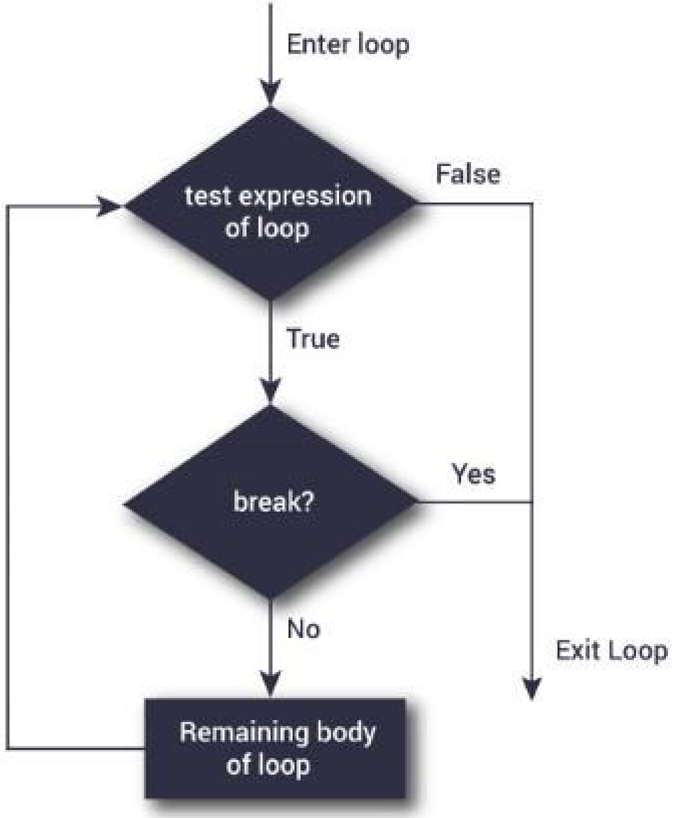
What's loop control statement?

"Changing the execution form its normal sequence"

Statements	Description
break statement	Terminates the loop statement
continue statement	Causes the loop to skip the remainder of its body

Loop Control Statements





while Loop

Syntax:

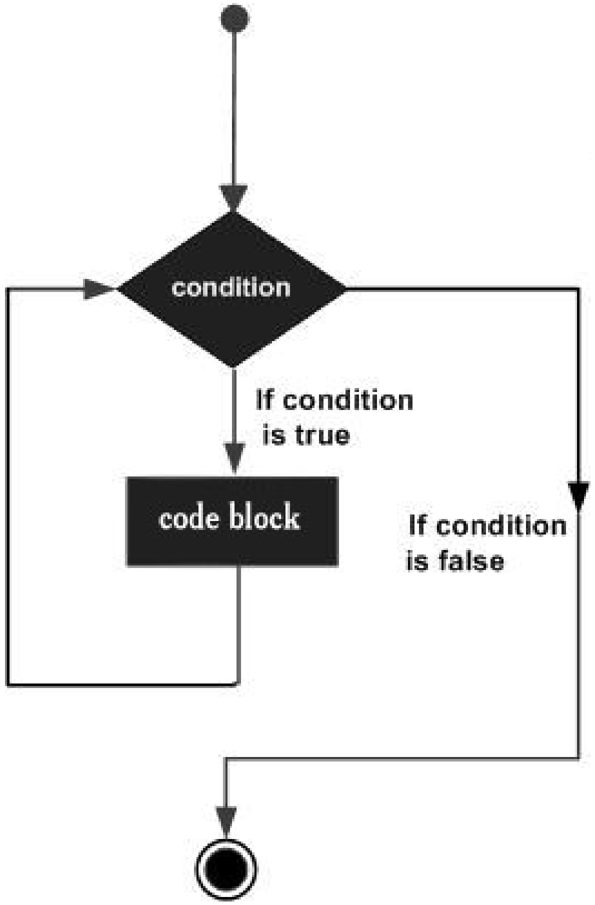
```
while (condition)
        statement(s);
end
```

Example:

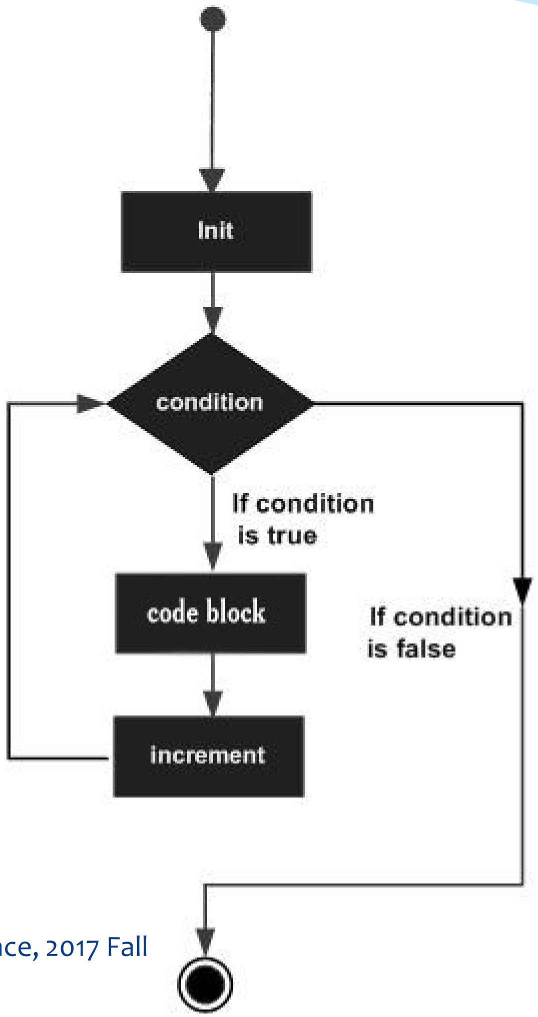
```
x = 10;
while (x < 20)
        disp(x);
        x = x + 1;
end
```

```
10
11
12
13
14
15
16
17
18
19
```

Note: while loop require a end keyword.



for Loop



Syntax:

```
for variable(s) = start value: increment : end value
          statement(s);
end
```

Example:

```
for x = 1 : 2 : 15
          disp(x);
end
```

```
1
3
5
7
9
11
13
15
```

Note: while loop require a end keyword.

nested Loop

Syntax 1:

```
while (condition)
        statement(s) or while statement(s);
end
```

Syntax 2:

```
for variable(s) = start value: increment : end value
        statement(s) or for statement(s);
end
```

MATLAB

Creating and Calling Functions

Functions

What's functions?

"A function is a group of statements that together perform a task."

Functions in MATLAB:

* Functions are defined in separate files.
* The name of the file and function's name should be the same.
* Functions can accept more than one input arguments and may return more than one output arguments.

Creating and Calling Functions

Syntax for creating:

```
function [output arguments] = function’s name(input arguments)
            Function’s body;
End
```

Syntax for calling:

```
function’s name(input arguments);
```

Creating and Calling Functions

Example for creating:

```
function [max_num] = arr_max(arr)
        max_num = arr(1);
        for i = 2 : size(arr, 2)
                if (arr(i) > max_num)
                        max_num = arr(i);
                end
        end
end
```

Example for calling:

```
x = [10 24 5 74 6];
disp(arr_max(x));
```

```
74
```

Function Types

- Primary Function
- Anonymous Function
- Sub-Functions
- Nested Functions
- Private Functions
- Global Variables

Primary Function

Syntax for creating:

```
function [output arguments] = function’s name(input arguments)
        Function’s body;
End
```

Syntax for calling:

```
function’s name(input arguments);
```

Anonymous Function

A anonymous function is like an inline

function in traditional programming languages, defined within a single MATLAB statement. It is a simple function without having to create s file.

Syntax for creating:

function's name = @(**arguments list**) expression

Syntax for calling:

function's name(**input arguments**);

Example:

```
power = @(x, y) x.^y
disp(power(7, 3));
```

```
343
```

Sub-Function

Syntax for creating:

```
function [output arguments] = function’s name(input arguments)
          statement(s) or you can locally use all sub-functions;
End
Sub-Function 1
Sub-Function 2
……….
Sub-Function n
```

Syntax for calling:

```
function’s name(input arguments);
```

Nested Functions

Syntax for creating:

```
function [output arguments] = function’s name(input arguments)
          Statement(s) or using nested functions locally;
          function [output arguments] = function’s name(input arguments)
                    Statement(s);
          end
End
```

Syntax for calling:

```
function’s name(input arguments);
```

Private Function

A private function is a primary function that is visible only to a limited group of other functions. Private functions reside in **subfolder** with the special name **private**.

Syntax for creating:

```
function [output arguments] = function's name(output arguments)
          Statement(s);
end
```

Syntax for calling in other function:

```
function's name(input arguments);
```

Note: you can use all variables of private function in calling functions.

Global Variables

Syntax for creating:

function [**output arguments**] = **function's name**(**input arguments**)

  Global **variables**

  Statement(s) with or without using global variables;

End

Syntax for calling:

Global **variables**;

function's name(**input arguments**);

MATLAB

Vectors, Matrix and Array

Vectors

What's vectors?

"A vector is a one-dimensional array of numbers."

- MATLAB allows to creating two types of **vectors**
  - **Row Vector** (with space)

    Syntax: vector's name = [item_1   item_2   ….   Item_n];

    Example: r = [3 8 12 8 24 55];
  - **Column Vector** (with semi-colon)

    Syntax: vector's name = [item_1;   item_2;   ….;   Item_n];

    Example: c = [3; 8; 12; 8; 24; 55];

Referencing the Elements of a Vector

- You can reference one or more of the elements of a vector in serval ways.
  - **Syntax for specific element:**

    vector's name(i^{th} $component\ of\ a\ vector$);

    Example: r = [3 8 12 8 24 55];

    disp(r(3));

    $\boxed{\textbf{12}}$
  - **Syntax for range elements:**

    vector's name(i^{th} $first : i^{th}$ $last$);

    Example: r = [3 8 12 8 24 55];

    disp(r(:));

    $\boxed{\textbf{3 8 12 8 24 55}}$

Deleting the Elements of a Vector

- You can delete one or more of the elements of a vector by assigning an empty set.
  - **Syntax for specific element:**

vector's name(i^{th} $component$) = [];

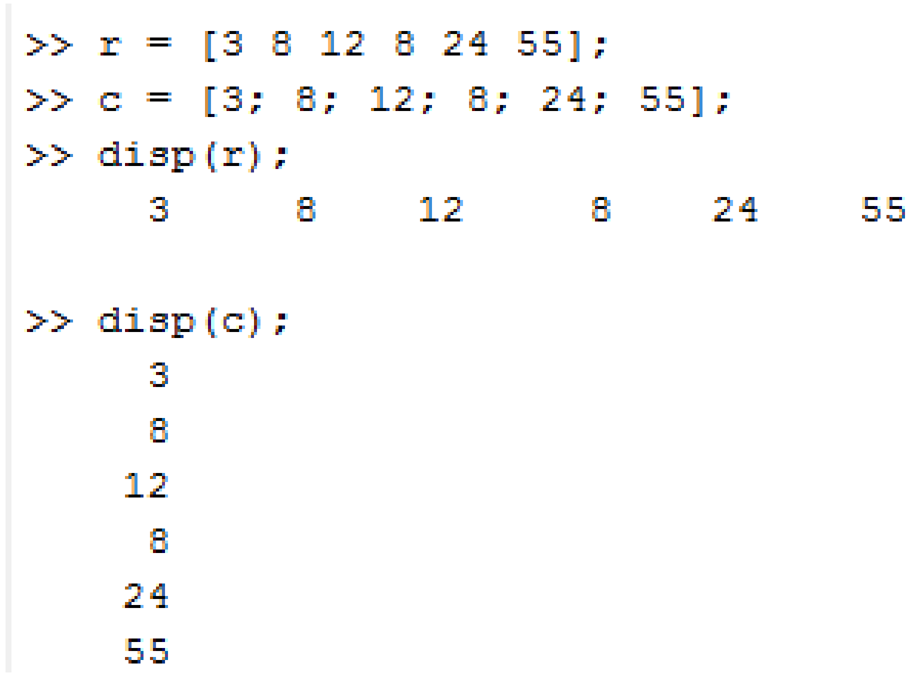
Example: arr = [3 7 12 8 24 55];

arr(4) = [];

$\boxed{3\ \ 7\ \ 12\ \ 24\ \ 55}$

Vectors

Example:



```
>> r = [3 8 12 8 24 55];
>> c = [3; 8; 12; 8; 24; 55];
>> disp(r);
     3     8    12     8    24    55

>> disp(c);
     3
     8
    12
     8
    24
    55
```

Matrix

What's matrix?

"A matrix is a two-dimensional array of numbers."

- MATLAB allows to create a matrix by entering elements in each row as comma or space and using semi-colons to mark the end of each row.
  - **Syntax:**

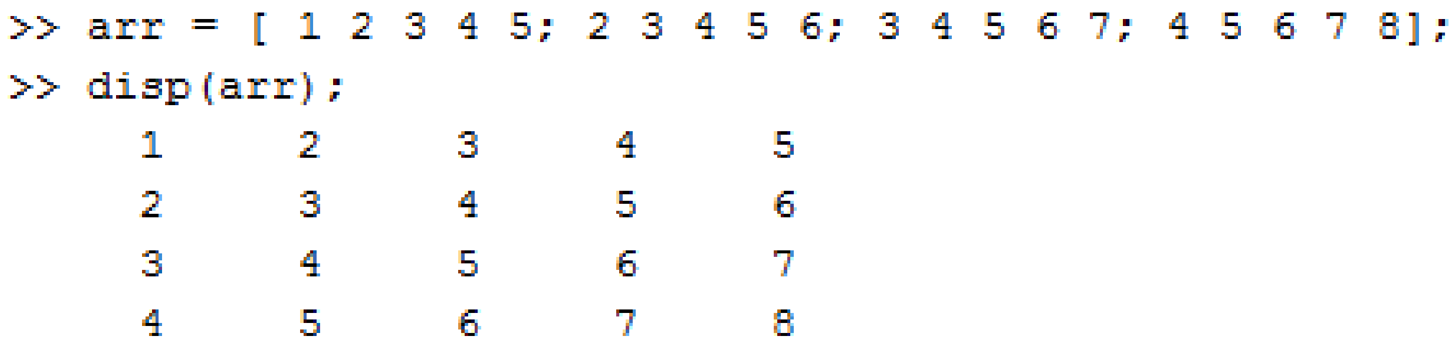
```
Matrix's name = [row_1_1  row_1_2  ….  row_1_n;
                 row_2_1   row_2_2  ….  row_2_n;
                 ……;
                 row_n_1  row_n_2  ….  row_n_n];
```

Example: Create a 4-by-5 matrix

```
arr = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
```

Matrix

Example:



```
>> arr = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
>> disp(arr);
     1     2     3     4     5
     2     3     4     5     6
     3     4     5     6     7
     4     5     6     7     8
```

Multidimensional Array

What's multidimensional array?

"An array having more than two dimensions."

- In MATLAB, multidimensional arrays are an extension of the normal two-dimensional matrix.
- All variables of all data types are multidimensional arrays.

Syntax 1:

- Create two-dimensional array.
- Extend this array.

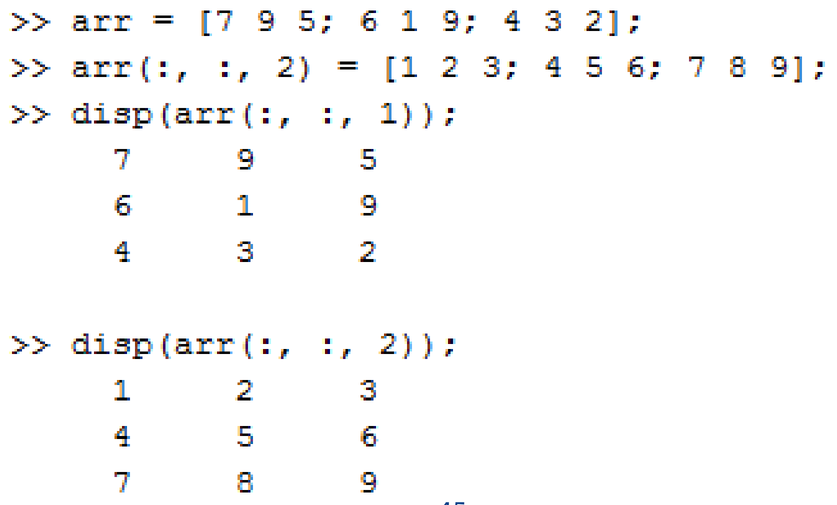
Multidimensional Array

Example:

- Create two-dimensional array.

  arr = [7 9 5; 6 1 9; 4 3 2];
- Extend this array.

  arr(:, :, 2) = [1 2 3; 4 5 6; 7 8 9];



```
>> arr = [7 9 5; 6 1 9; 4 3 2];
>> arr(:, :, 2) = [1 2 3; 4 5 6; 7 8 9];
>> disp(arr(:, :, 1));
     7     9     5
     6     1     9
     4     3     2

>> disp(arr(:, :, 2));
     1     2     3
     4     5     6
     7     8     9
```

Multidimensional Array

Syntax 2:

- using **cat** function to build multidimensional arrays

array's name = **cat**(dimension, array 1, array 2, …., array n);

Example:

arr1 = [9 8 7; 6 5 4; 3 2 1];

arr2 = [1 2 3; 4 5 6; 7 8 9];

arr3 = **cat**(3, arr1, arr2, [2 3 1; 4 7 8; 3 9 8]);

Referencing the Elements of a Matrix and Multidimensional array

- You can reference one or more of the elements of a vector in serval ways.
  - **Syntax for specific element:**

    vector's name($i^{th}\ row,\ i^{th}\ column$);

    Example: arr = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

    disp(r(2, 5));

```
6
```

  - **Syntax for range elements:**

    vector's name($i^{th}\ first : \ i^{th}\ last$);

    Example: arr = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

    disp(r(3, :));

```
3
4
5
6
7
```

Referencing the Elements of a Matrix and Multidimensional array

* **Syntax for range elements:**

  vector's name($i^{th}\ first : \ i^{th}\ last$);

  Example: arr = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

  disp(r(:, 2:3));

```
2    3
3    4
4    5
5    6
```

Referencing the Elements of a Matrix and Multidimensional array

- **Syntax for range elements:**

  vector's name($i^{th}\ first:\ i^{th}\ last$);

  Example: arr = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

  disp(r(2:3, 2:4));

```
3     4     5
4     5     6
```

Deleting a Row or a Column in Matrix and Multidimensional array

- You can delete an entire row or column of a matrix by assigning an empty set.
  - **Syntax for specific element or range:**

vector's name(i^{th} *row*, i^{th} *column*) = [];

Example: arr = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

arr(4, :) = [];

$$\boxed{\begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \end{array}}$$

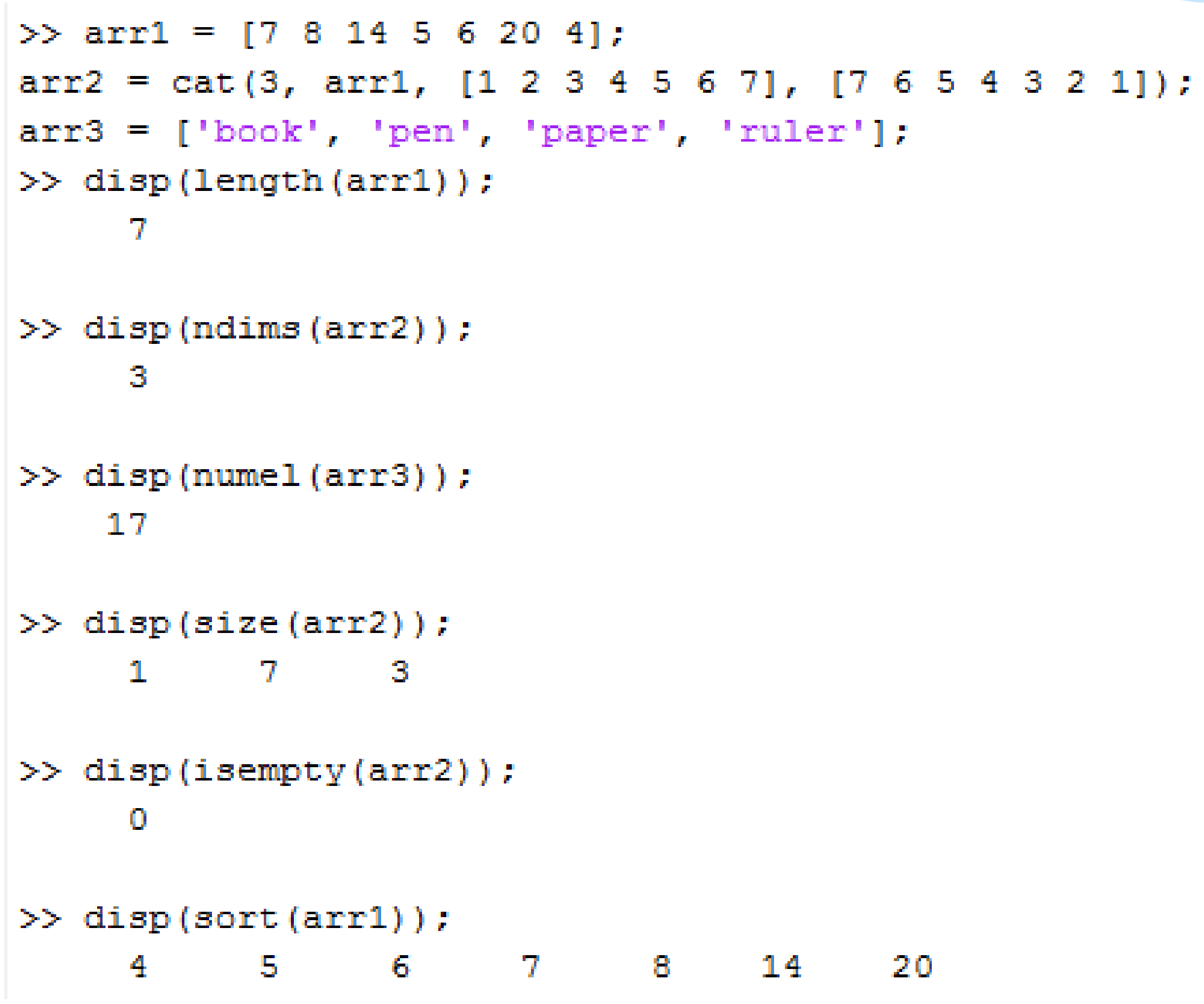
Array Functions

functions	Description
length	Length of vector or largest array dimension
ndims	Number of array dimensions
numel	Number of array elements
size	Array dimensions
iscolumn	Determines whether input is column vector
isempty	Determines whether input is empty
ismatrix	Determines whether input is matrix
isrow	Determines whether input is row vector
isvector	Determines whether input is vector

Array Functions

functions	Description
fliplr	Flips matrix from left to right
flipud	Flips matrix from up to down
rot90	Rotates matrix 90 degrees
sort	Sorts array elements in ascending or descending order
transpose	Transponse

Array Functions Examples



```
>> arr1 = [7 8 14 5 6 20 4];
arr2 = cat(3, arr1, [1 2 3 4 5 6 7], [7 6 5 4 3 2 1]);
arr3 = ['book', 'pen', 'paper', 'ruler'];
>> disp(length(arr1));
     7

>> disp(ndims(arr2));
     3

>> disp(numel(arr3));
    17

>> disp(size(arr2));
     1     7     3

>> disp(isempty(arr2));
     0

>> disp(sort(arr1));
     4     5     6     7     8    14    20
```

MATLAB

Graphics

Bar

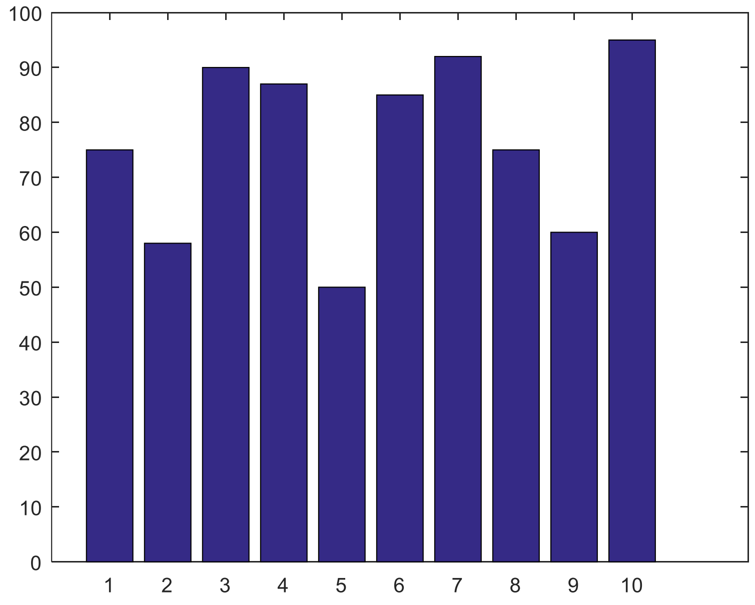
- The **bar** command draws a two-dimensional bar chart.

  Syntax:

  ```
  bar(First-dimension, Second-dimension);
  ```

  Example:

  ```
  arr1 = [1:10];
  arr2 = [75, 58, 90, 87, 50, 85,
          92, 75, 60, 95];
  bar(arr1, arr2);
  ```



Graphics in 2D and 3D

- **Graphics** functions include 2D and 3D plotting functions to visualize data and communicate results. For example, you can compare sets of data, track changes in data over time, or show data distribution.
- There are two basic graphical functions:
  - **Plot** for 2D graph
  - **Surf** for 3D graph

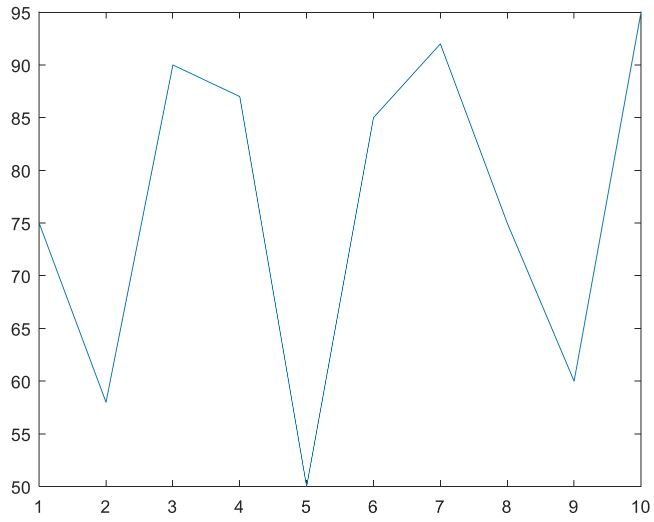
Plotting for 2D

Syntax:

```
plot(First-dimension, Second-dimension);
```

Example:

```
arr1 = [1:10];
arr2 = [75, 58, 90, 87, 50, 85,
        92, 75, 60, 95];
plot(arr1, arr2);
```



surf for 3D

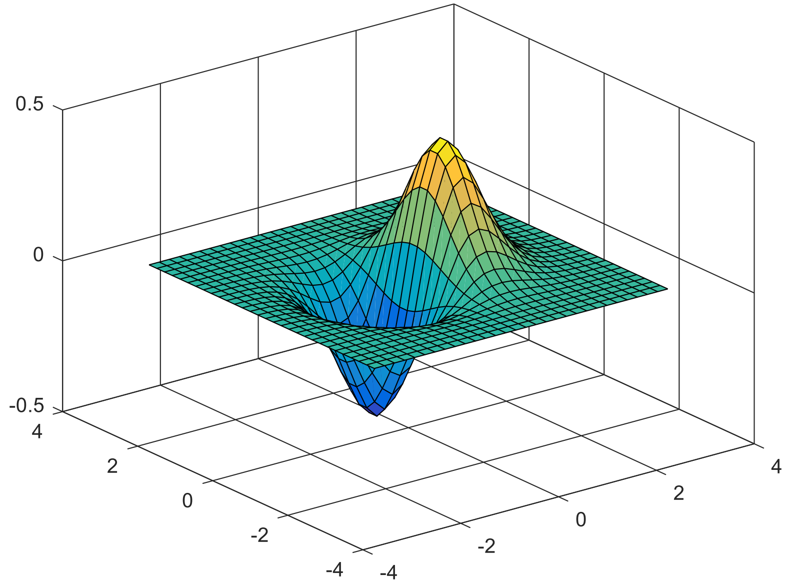
Syntax:

surf(First-dimension, Second-D, Third-D);

Example: create a 3D surface map for the function

$$g = xe^{-(x^2+y^2)}$$

```
[x, y] = peaks(30);
z = x .* exp(-x.^2 - y.^2);
surf(x, y, z);
```



Summary

We learn:

- Control Flow
  - Decision Making
  - Loop Types
- Creating and Calling a New Function
- Vector, Matrix and Array
- Graphics
  - Bar
  - Graphics in 2D and 3D

Note that it is brief in MATLAB concepts.