Scan or click here for more resources

MATLAB if...end statement

- The **if** is a conditional statement that provides the functionality to choose a block of code to execute at run time.
- A predefined condition is checked, and the remaining code executes based on the output of the condition.
- The **if** statement is defined by the **if**
- The control flows within the **if** block if the condition is found true.
- Always close the if block with the **end**
- The **elseif** and **else** are optional, use them if there are more conditions to be checked.
- The **elseif** and **else** are an optional parts of the **if** statement and don't require additional **end**
- Remember not to use space in between else & if of the **elseif** keyword, because this leads to nested **if** statement and each **if** block needs to be close with the **end**
- There is no limit to use multiple **elseif**
- We can nest the **if** statement by using it within another **if**

Example

```
% program to announce bonus on max hour of job
a = randi(250,12,2)
pr = 0;
bonus = 10000;
for k = 1:numel(a)
    pr = pr + a(k);
end
disp(['total hours of job done by you in a year are: ',num2str(pr)])
if pr >= 2800
    disp(['Great...you earned a bonus amount of Rs. ',num2str(bonus)])
```

```
11.     else
12.         disp(['better luck next year....'])
13.     end
```

MATLAB if-else... end statement

If the first condition is not true, then we can define other statements to run by using the **else** keyword.

Syntax:

```
1.      if expression
2.          Statements
3.      else
4.          Statements
5.      end
```

Example1:

```
1.      % program to check the number is even or odd
2.      a = randi(100,1);
3.      if rem(a,2) == 0
4.          disp('an even number')
5.      else
6.          disp('an odd number')
7.      end
```

MATLAB if-elseif-else...end statement

If we have more than one option or condition to check, then use **elseif** keyword.

Syntax:

```
1.      if expression
2.              Statements
3.      elseif expression
4.       Statements
5.      else
6.              Statements
7.      end
```

Example:

```
1.      % program to compare two numbers
2.      % generate random number for your age
3.      n = randi(100,1);
4.      age = 23;
```

```
% check the number is greater than your age
if n > age
   disp('i am younger')
elseif n < age
   disp('you are younger')
else
   disp('we are of same age')
end
```

MATLAB nested if-else

- If statements can be nested, but each if statement requires the **end** keyword.

Syntax:

```
if expression
   Statements
   if expression
      Statements
   else
      Statements
   end
elseif expression
   Statements
   if expression
      Statements
   end
else
   Statements
end
```

Example1

```
% nested if-elseif-else program
num = randi(100,1);
a = input('enter a number greater than 1 and less than 10 : ')
disp(['the random number is : ', num2str(num)])
if a <= 1 || a >= 10
   disp('please try again & enter the number in between 1 & 10')
elseif rem(num,a) == 0
      if rem(num,2) == 0
         if rem(a,2) == 0
            disp('you entered an even number, and the random number is also even')
```

```
            else
                disp('random number is even and divisible by the entered number')
            end
        else
            disp('random number is odd but divisible by the entered number')
        end
    elseif num < a
        disp('random number is smaller than the  entered number, please try again ')
    else
        disp('random number is not divisible by the entered number')
    end
```

MATLAB switch

The switch is another type of conditional statement and executes one of the group of several statements.

- If we want to test the equality against a pre-defined set of rules, then the switch statement can be an alternative of the if statement.

Syntax:

```
switch switch_expression
    case case_expression1
        Statements
    case case_expression2
        Statements
    case case_expressionN
        Statements
    otherwise
        Statements
End
```

MATLAB Loops

A loop statement allow us to execute a statement or group of statements multiple times.

MATLAB provides different types of loops to handle looping requirements, including while loops, for loops, and nested loops. If we are trying to declare or write our own loops, we need to make sure that the loops are written as scripts and not directly in the Command Window.

Two additional command, **break and continue**, can be used to create a third type of loop, known as **midpoint break loop**. Midpoint break loop is useful for situations where the commands in the loop must be executed at least once, but where the decision to exit the loop is based on some criterion.

Types of Loops

There are two types of loop in MATLAB.

1. for
2. While

for loop

A for loop is used to repeat a statement or a group of statements for a fixed number of times.The for loop is used to loop the statements a specific number of times. And it also keeps track of each iteration with an incrementing or decrementing index variable.

Syntax

```
for index = values
 <program statements>
 ...
end
```

Example1:

```
% program to print multiples of first prime number between 1000 and 2000
% using for loop
pr = 0;
for k = 1000:2000
   if isprime(k)
      pr = k;
      disp(['The first prime number is : ',num2str(pr)])
      for m = pr:pr:pr*10
         disp(m)
      end

      break
   end
end
```

while loop

A while loop is used to execute a statement or a group of statements for an indefinite number of times until the conditional specified by while is no longer satisfied.The while loop repeatedly executes statements while a specified statement is true.

Syntax

```
 while <expression>
```

```
    <statements>
end
```

Example1:

```
% program to find the number ten from a series of random numbers
% using while loop
k = 1;
while k
   if randi(50,1) == 10
      disp(['The random number equivalent to 10 found at ',num2str(k),' step'])
      break
   end
   k = k + 1;
end
```

MATLAB Nested Loop

MATLAB also allows using one loop inside another loops.

The syntax for the nested for loop statement in MATLAB is as follows:

```
for m = 1:j
for n = 1:k
 <statements>;

     end
end
```

The syntax for the nested while loop statement in MATLAB is as follows:

```
while <expression1>
 while <expression2>
    <statements>
    end
end
```

Example:

We can use the nested for loop to display all the prime numbers from 1 to 100.

```
for i=2:100
      for j=2:100
         if(~mod(i, j))
            break; % if factor found, not prime
         end
```

```
        end
        if(j > (i/j))
            fprintf('%d is prime\n', i);
        end
    end
```

MATLAB break

The break statement terminate the execution of a for loop or while loop. When a break statement is encountered, execution proceeds with the next statement outside of the loop. In nested loops, break exists from the innermost loop only.

Example1:

```
% program to break the flow at a specified point

a = randi(100,6,6)
k = 1;
while k
   disp('program running smoothly')
   if a(k) == 27
      disp('program encounters the number 27, which is not useful for the current program;')
      disp(['at index no.:',num2str(k)])
      disp('so loop terminates now.....bye bye')
      break
   end
   k = k+1;
end
```

MATLAB continue:

The continue statement works within a for or while loop and passes control to the next iteration of the loop.

Example1:

```
% program to print all numbers divisible by 3 and skip remaining
a = (1:4:50); % creates row vector from 1 to 50 with a step of 4
for k = 1:numel(a)
   if rem(a(k),3)
      continue
   end
   disp(a(k))
end
```

MATLAB Error Control Statement-try, catch

MATLAB define some functions that are used to control error. The try-catch statement is an error control function, which is explained below.

Try - catch statement

Try-catch statement provides error handling control. General form of the try-catch statement is

Syntax:

```
try
    Statements
catch   exception
    Statements
end
```

Statements between try and catch execute first. If no error appears in executing statements between try and catch, MATLAB further executes the statements/code after the end keyword. If an error occurs during the execution of statements between try and catch, MATLAB executes statements between catch and end. Try-catch statement can be explained with the help of the following example.

Example:

```
a = ones(4);
b = zeros(3);
try
   c = [a;b];
catch ME
   disp(ME)
end
```

MATLAB - Functions

A function is a group of statements that together perform a task. In MATLAB, functions are defined in separate files. The name of the file and of the function should be the same.

Functions operate on variables within their own workspace, which is also called the **local workspace**, separate from the workspace you access at the MATLAB command prompt which is called the **base workspace**.

Functions can accept more than one input arguments and may return more than one output arguments.

Syntax of a function statement is –

```
function [out1,out2, ..., outN] = myfun(in1,in2,in3, ..., inN)
```

Example

The following function named *mymax* should be written in a file named *mymax.m*. It takes five numbers as argument and returns the maximum of the numbers.

Create a function file, named mymax.m and type the following code in it –

```
function max = mymax(n1, n2, n3, n4, n5)
%This function calculates the maximum of the% five numbers given as input
max =  n1;if(n2 > max)
   max = n2;endif(n3 > max)
   max = n3;endif(n4 > max)
   max = n4;endif(n5 > max)
   max = n5;end
```

The first line of a function starts with the keyword **function**. It gives the name of the function and order of arguments. In our example, the *mymax* function has five input arguments and one output argument.

The comment lines that come right after the function statement provide the help text. These lines are printed when you type –

```
help mymax
```

MATLAB will execute the above statement and return the following result –

```
This function calculates the maximum of the
   five numbers given as input
```

You can call the function as –

```
mymax(34, 78, 89, 23, 11)
```

MATLAB will execute the above statement and return the following result –

```
ans = 89
```

Anonymous Functions

An anonymous function is like an inline function in traditional programming languages, defined within a single MATLAB statement. It consists of a single MATLAB expression and any number of input and output arguments.

You can define an anonymous function right at the MATLAB command line or within a function or script.

This way you can create simple functions without having to create a file for them.

The syntax for creating an anonymous function from an expression is

```
f = @(arglist)expression
```

Example

In this example, we will write an anonymous function named power, which will take two numbers as input and return first number raised to the power of the second number.

Create a script file and type the following code in it –

Live Demo

```
power = @(x, n) x.^n;
result1 = power(7, 3)
result2 = power(49, 0.5)
result3 = power(10, -10)
result4 = power (4.5, 1.5)
```

When you run the file, it displays –

```
result1 =  343
result2 =  7
result3 =  1.0000e-10
result4 =  9.5459
```

Primary and Sub-Functions

Any function other than an anonymous function must be defined within a file. Each function file contains a required primary function that appears first and any number of optional sub-functions that comes after the primary function and used by it.

Primary functions can be called from outside of the file that defines them, either from command line or from other functions, but sub-functions cannot be called from command line or other functions, outside the function file.

Sub-functions are visible only to the primary function and other sub-functions within the function file that defines them.

Example

Let us write a function named quadratic that would calculate the roots of a quadratic equation. The function would take three inputs, the quadratic co-efficient, the linear co-efficient and the constant term. It would return the roots.

The function file quadratic.m will contain the primary function *quadratic* and the sub-function *disc*, which calculates the discriminant.

Create a function file *quadratic.m* and type the following code in it –

```
function [x1,x2] = quadratic(a,b,c)
%this function returns the roots of % a quadratic equation.% It takes 3 input arguments% which are the co-
efficients of x2, x and the %constant term% It returns the roots
d = disc(a,b,c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);end    % end of quadratic
function dis = disc(a,b,c) %function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);end    % end of sub-function
```

You can call the above function from command prompt as –

```
quadratic(2,4,-4)
```

MATLAB will execute the above statement and return the following result –

```
ans = 0.7321
```

Nested Functions

You can define functions within the body of another function. These are called nested functions. A nested function contains any or all of the components of any other function.

Nested functions are defined within the scope of another function and they share access to the containing function's workspace.

A nested function follows the following syntax –

```
function x = A(p1, p2)
...
B(p2)
   function y = B(p3)
   ...
   end
...
end
```

Example

Let us rewrite the function *quadratic*, from previous example, however, this time the disc function will be a nested function.

Create a function file *quadratic2.m* and type the following code in it –

```
function [x1,x2] = quadratic2(a,b,c)function disc   % nested function
d = sqrt(b^2 - 4*a*c);end    % end of function disc

disc;
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);end    % end of function quadratic2
```

You can call the above function from command prompt as –

```
quadratic2(2,4,-4)
```

MATLAB will execute the above statement and return the following result –

```
ans =  0.73205
```

Private Functions

A private function is a primary function that is visible only to a limited group of other functions. If you do not want to expose the implementation of a function(s), you can create them as private functions.

Private functions reside in **subfolders** with the special name **private**.

They are visible only to functions in the parent folder.

Example

Let us rewrite the *quadratic* function. This time, however, the *disc* function calculating the discriminant, will be a private function.

Create a subfolder named private in working directory. Store the following function file *disc.m* in it –

```
function dis = disc(a,b,c) %function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);end       % end of sub-function
```

Create a function quadratic3.m in your working directory and type the following code in it –

```
function [x1,x2] = quadratic3(a,b,c)
%this function returns the roots of % a quadratic equation.% It takes 3 input arguments% which are the co-
efficient of x2, x and the %constant term% It returns the roots
d = disc(a,b,c);

x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);end       % end of quadratic3
```

You can call the above function from command prompt as –

```
quadratic3(2,4,-4)
```

MATLAB will execute the above statement and return the following result –

```
ans =  0.73205
```

Global Variables

Global variables can be shared by more than one function. For this, you need to declare the variable as global in all the functions.

If you want to access that variable from the base workspace, then declare the variable at the command line.

The global declaration must occur before the variable is actually used in a function. It is a good practice to use capital letters for the names of global variables to distinguish them from other variables.

Example

Let us create a function file named average.m and type the following code in it –

```
function avg = average(nums)global TOTAL
avg = sum(nums)/TOTAL;end
```

Create a script file and type the following code in it –

```
global TOTAL;
TOTAL = 10;
n = [34, 45, 25, 45, 33, 19, 40, 34, 38, 42];
av = average(n)
```