SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB

Unit 6-Lecture 22

"if ... end" structure

BTech (CSBS) -Semester VII

4 October 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

Control Flow and Operators

1) relational and logical operators
2) “if ... end" structure
3) “for ... end" loop
4) “while ... end" loop
5) other flow structures
6) operator precedence
7) saving output to a file

The “IF” statement

The **if** statement chooses whether another statement, or group of statements, is executed or not. The general form of the **if** statement is:

```
if condition
    action
end
```

For example, the following **if** statement checks to see whether the value of a variable is negative. If it is, the value is changed to a zero; otherwise, nothing is changed.

```
if num < 0
     num = 0
end
```

The “IF” statement: Example

sqrtifexamp.m

```
% Prompt the user for a number and print its sqrt

num = input('Please enter a number: ');

% If the user entered a negative number, change it
if num < 0
    num = 0;
end
fprintf('The sqrt of %.1f is %.1f\n',num,sqrt(num))
```

Here are two examples of running this script:

```
>> sqrtifexamp
Please enter a number: -4.2
The sqrt of 0.0 is 0.0

>> sqrtifexamp
Please enter a number: 1.44
The sqrt of 1.4 is 1.2
```

The “IF” statement: Example

sqrtifexampii.m

```
% Prompt the user for a number and print its sqrt

num = input('Please enter a number: ');

% If the user entered a negative number, tell
% the user and change it
if num < 0
    disp('OK, we''ll use the absolute value')
    num = abs(num);
end
fprintf('The sqrt of %.1f is %.1f\n',num,sqrt(num))
```

```
>> sqrtifexampii
Please enter a number: -25
OK, we'll use the absolute value
The sqrt of 25.0 is 5.0
```

The "IF" statement: Question

Assume that we want to create a vector of increasing integer values from *mymin* to *mymax*. We will write a function *createvec* that receives two input arguments, *mymin* and *mymax*, and returns a vector with values from *mymin* to *mymax* in steps of one. First, we would make sure that the value of *mymin* is less than the value of *mymax*. If not, we would need to exchange their values before creating the vector. How would we accomplish this?

Answer: To exchange values, a third variable, a temporary variable, is required. For example, let's say that we have two variables, *a* and *b*, storing the values:

```
a = 3;
b = 5;
```

To exchange values, we could *not* just assign the value of *b* to *a*, as follows:

```
a = b;
```

If that were done, then the value of *a* (the 3), is lost! Instead, we need to assign the value of *a* first to a ***temporary variable*** so that the value is not lost. The algorithm would be:

- assign the value of *a* to *temp*
- assign the value of *b* to *a*
- assign the value of *temp* to *b*

```
>> temp = a;
>> a = b
a =
     5
>> b = temp
b =
     3
```

Now, for the function. An **if** statement is used to determine whether or not the exchange is necessary.

The "IF" statement: Question

createvec.m

```
function outvec = createvec(mymin, mymax)
% createvec creates a vector that iterates from a
%  specified minimum to a maximum
% Format of call: createvec(minimum, maximum)
% Returns a vector

%If the "minimum" isn't smaller than the "maximum",
% exchange the values using a temporary variable
if mymin > mymax
    temp = mymin;
    mymin = mymax;
    mymax = temp;
end

% Use the colon operator to create the vector
outvec = mymin:mymax;
end
```

Examples of calling the function are:

```
>> createvec(4,6)
ans =
     4   5   6

>> createvec(7,3)
ans =
     3   4   5   6   7
```

"if-else" Statement

The **if** statement chooses whether or not an action is executed. Choosing between two actions, or choosing from among several actions, is accomplished using **if-else**, nested **if-else**, and **switch** statements.

The **if-else** statement is used to choose between two statements, or sets of statements. The general form is:

```
if condition
    action1
else
    action2
end
```

“if-else” Statement: Example

For example, to determine and print whether or not a random number in the range from 0 to 1 is less than 0.5, an **if-else** statement could be used:

```
if rand < 0.5
   disp('It was less than .5!')
else
   disp('It was not less than .5!')
end
```

"if-else" Statement: Example

One application of an **if-else** statement is to check for errors in the inputs to a script (this is called ***error-checking***). For example, an earlier script prompted the user for a radius, and then used that to calculate the area of a circle. However, it did not check to make sure that the radius was valid (e.g., a positive number). Here is a modified script that checks the radius:

checkradius.m

```
% This script calculates the area of a circle
% It error-checks the user's radius
radius = input('Please enter the radius: ');
if radius <= 0
    fprintf('Sorry; %.2f is not a valid radius\n',radius)
else
    area = calcarea(radius);
    fprintf('For a circle with a radius of %.2f,',radius)
    fprintf(' the area is %.2f\n',area)
end
```

“if-else” Statement: Example

Examples of running this script when the user enters invalid and then valid radii are shown as follows:

```
>> checkradius
Please enter the radius: -4
Sorry; -4.00 is not a valid radius

>> checkradius
Please enter the radius: 5.5
For a circle with a radius of 5.50, the area is 95.03
```

```
>> if radius <= 0
   error('Sorry; %.2f is not a valid radius\n', radius)
end

Sorry; -4.00 is not a valid radius
```

"if-else" Statement: Example

Examples of running this script when the user enters invalid and then valid radii are shown as follows:

```
>> checkradius
Please enter the radius: -4
Sorry; -4.00 is not a valid radius

>> checkradius
Please enter the radius: 5.5
For a circle with a radius of 5.50, the area is 95.03
```

```
>> if radius <= 0
   error('Sorry; %.2f is not a valid radius\n', radius)
end

Sorry; -4.00 is not a valid radius
```

Nested “if-else” Statement

$y = 1$ if $x < -1$
$y = x^2$ if $-1 \leq x \leq 2$
$y = 4$ if $x > 2$

The value of y is based on the value of x, which could be in one of three possible ranges. Choosing which range could be accomplished with three separate if statements, is as follows:

```
if x < -1
   y = 1;
end
if x >= -1 && x <=2
   y = x^2;
end
if x > 2
   y = 4;
end
```

Nested “if-else” Statement

Above example could also be written as below:

```
if x < -1
   y = 1;
else
   % If we are here, x must be >= -1
   % Use an if-else statement to choose
   %   between the two remaining ranges
   if  x <= 2
       y = x^2;
   else
       % No need to check
       % If we are here, x must be > 2
       y = 4;
   end
end
```

Nested “if-else” Statement

THE PROGRAMMING CONCEPT

In some programming languages, choosing from multiple options means using nested **if-else** statements. However, MATLAB has another method of accomplishing this using the **elseif** clause.

THE EFFICIENT METHOD

To choose from among more than two actions, the **elseif** clause is used. For example, if there are n choices (where $n > 3$ in this example), the following general form would be used:

Nested “if-else” Statement

THE EFFICIENT METHOD—CONT’D

```
if condition1
    action1
elseif condition2
    action2
elseif condition3
     action3
% etc: there can be many of these
else
    actionn     % the nth action
end
```

The actions of the **if**, **elseif**, and **else** clauses are naturally bracketed by the reserved words **if**, **elseif**, **else**, and **end**.

"elseif" Statement

For example, the previous example could be written using the **elseif** clause, rather than nesting **if-else** statements:

```
if x < -1
   y = 1;
elseif x <= 2
   y = x^2;
else
   y = 4;
end
```

"elseif" Statement

This could be implemented in a function that receives a value of x and returns the corresponding value of y:

calcy.m

```
function y = calcy(x)
% calcy calculates y as a function of x
% Format of call: calcy(x)
% y = 1          if    x < -1
% y = x^2        if    -1 <= x <= 2
% y = 4          if    x > 2

if x < -1
    y = 1;
elseif x <= 2
    y = x^2;
else
    y = 4;
end
end
```

```
>> x = 1.1;
>> y = calcy(x)
y =
   1.2100
```

"elseif" Statement

How could you write a function to determine whether an input argument is a scalar, a vector, or a matrix?

Answer: To do this, the **size** function can be used to find the dimensions of the input argument. If both the number of rows and columns is equal to 1, then the input argument is scalar. If, however, only one dimension is 1, the input argument is a vector (either a row or column vector). If neither dimension is 1, the input argument is a matrix. These three options can be tested using a nested **if-else** statement. In this example, the word 'scalar,' 'vector,' or 'matrix' is returned from the function.

Note that there is no need to check for the last case: if the input argument isn't a scalar or a vector, it must be a matrix! Examples of calling this function are:

```
>> findargtype(33)
ans =
scalar

>> disp(findargtype(2:5))
vector

>> findargtype(zeros(2,3))
ans =
matrix
```

findargtype.m

```
function outtype = findargtype(inputarg)
% findargtype determines whether the input
%    argument is a scalar, vector, or matrix
% Format of call: findargtype(inputArgument)
% Returns a string

[r c] = size(inputarg);
if r ==1 && c ==1
   outtype = 'scalar';
elseif r ==1 || c ==1
    outtype = 'vector';
else
    outtype = 'matrix';
end
end
```

"elseif" Statement

letgrade.m

```
function grade = letgrade(quiz)
% letgrade returns the letter grade corresponding
%    to the integer quiz grade argument
% Format of call: letgrade(integerQuiz)
% Returns a character

% First, error-check
if quiz < 0 || quiz > 10
    grade = 'X';

% If here, it is valid so figure out the
%  corresponding letter grade
elseif quiz == 9 || quiz == 10
     grade = 'A';
elseif quiz == 8
     grade = 'B';
elseif quiz == 7
     grade = 'C';
elseif quiz == 6
     grade = 'D';
else
     grade = 'F';
end
end
```

Three examples of calling this function are:

```
>> quiz =8;
>> lettergrade = letgrade(quiz)
lettergrade =
B

>> quiz =4;
>> letgrade(quiz)
ans =
F

>> lg = letgrade(22)
lg =
X
```

Practise question

PRACTICE 4.1

Write an **if** statement that would print "Hey, you get overtime!" if the value of a variable *hours* is greater than 40. Test the **if** statement for values of *hours* less than, equal to, and greater than 40. Will it be easier to do this in the Command Window or in a script?

PRACTICE 4.3

Modify the function *findargtype* to return either 'scalar,' 'row vector,' 'column vector,' or 'matrix,' depending on the input argument.

PRACTICE 4.4

Modify the original function *findargtype* to use three separate **if** statements instead of a nested **if-else** statement.

Practise question

PRACTICE 4.2

Write a script *printsindegorrad* that will:

- prompt the user for an angle
- prompt the user for (r)adians or (d)egrees, with radians as the default
- if the user enters 'd,' the **sind** function will be used to get the sine of the angle in degrees; otherwise, the **sin** function will be used. Which sine function to use will be based solely on whether the user entered a 'd' or not ('d' means degrees, so **sind** is used; otherwise, for any other character the default of radians is assumed so **sin** is used)
- print the result.

Here are examples of running the script:

```
>> printsindegorrad
Enter the angle: 45
(r)adians (the default) or (d)egrees: d
The sin is 0.71

>> printsindegorrad
Enter the angle: pi
(r)adians (the default) or (d)egrees: r
The sin is 0.00
```