CHAPTER 4

Selection Statements

KEY TERMS

selection statements	action	nesting statements
branching statements	temporary variable	cascading if-else
condition	error-checking	"is" functions

CONTENTS

In the scripts and functions we've seen thus far, every statement was executed in sequence. That is not always desirable, and in this chapter we'll see how to make choices as to whether statements are executed or not, and how to choose between or among statements. The statements that accomplish this are called ***selection*** or ***branching*** statements.

The MATLAB® software has two basic statements that allow us to make choices: the **if** statement and the **switch** statement. The **if** statement has optional **else** and **elseif** clauses for branching. The **if** statement uses expressions that are logically **true** or **false**. These expressions use relational and logical operators. MATLAB also has a **menu** function that presents choices to the user; this will be covered at the end of this chapter.

4.1 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
```

A ***condition*** is a relational expression that is conceptually, or logically, **true** or **false**. The ***action*** is a statement, or a group of statements, that will be executed if the condition is **true**. When the **if** statement is executed, first the condition is

MATLAB®. http://dx.doi.org/10.1016/B978-0-12-405876-7.00004-3
Copyright © 2013 Elsevier Inc. All rights reserved.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

evaluated. If the value of the condition is **true**, the action will be executed; if not, the action will not be executed. The action can be any number of statements until the reserved word **end**; the action is naturally bracketed by the reserved words **if** and **end**. (Note that this is different from the **end** that is used as an index into a vector or matrix.) The action is usually indented to make it easier to see.

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
```

If statements can be entered in the Command Window, although they generally make more sense in scripts or functions. In the Command Window, the **if** line would be entered, followed by the Enter key, the action, the Enter key, and, finally, **end** and Enter. The results will follow immediately. For example, the preceding **if** statement is shown twice here.

```
>> num = -4;
>> if num < 0
       num = 0
   end
num =
     0

>> num = 5;
>> if num < 0
       num = 0
   end
>>
```

Note that the output from the assignment is not suppressed, so the result of the action will be shown if the action is executed. The first time the value of the variable is negative so the action is executed and the variable is modified, but, in the second case, the variable is positive so the action is skipped.

This may be used, for example, to make sure that the square root function is not used on a negative number. The following script prompts the user for a number and prints the square root. If the user enters a negative number the **if** statement changes it to zero before taking the square root.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

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
```

Note that in the script the output from the assignment statement is suppressed. In this case, the action of the **if** statement was a single assignment statement. The action can be any number of valid statements. For example, we may wish to print a note to the user to say that the number entered was being changed. Also, instead of changing it to zero we will use the absolute value of the negative number entered by the user.

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
```

Note that, as seen in this example, two single quotes in the **disp** statement are used to print one single quote.

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?

cuu duong than cong . com

QUICK 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.

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
```

CuuDuongThanCong.com https://fb.com/tailieudientucntt

4.1.1 Representing Logical True and False

It has been stated that conceptually true expressions have the **logical** value of 1 and expressions that are conceptually false have the **logical** value of 0. Representing the concepts of **logical true** and **false** in MATLAB is slightly different: the concept of false is represented by the value of 0, but the concept of true can be represented by *any nonzero value* (not just 1). This can lead to some strange **logical** expressions. For example:

```
>> all(1:3)
ans =
     1
```

Also, consider the following **if** statement:

```
>> if 5
       disp('Yes, this is true!')
   end
Yes, this is true!
```

As 5 is a nonzero value, the condition is **true**. Therefore, when this **logical** expression is evaluated, it will be **true**, so the **disp** function will be executed and "Yes, this is true" is displayed. Of course, this is a pretty bizarre **if** statement — one that hopefully would never be encountered!

However, a simple mistake in an expression can lead to a similar result. For example, let's say that the user is prompted for a choice of 'Y' or 'N' for a yes/no question.

```
letter = input('Choice (Y/N): ','s');
```

In a script we might want to execute a particular action if the user responded with 'Y'. Most scripts would allow the user to enter either lowercase or uppercase; for example, either 'y' or 'Y' to indicate "yes". The proper expression that would return **true** if the value of *letter* was 'y' or 'Y' would be

```
letter == 'y' || letter == 'Y'
```

However, if by mistake this was written as:

```
letter == 'y' || 'Y'   %Note: incorrect!!
```

this expression would ALWAYS be **true**, regardless of the value of the variable *letter*. This is because `'Y'` is a nonzero value, so it is a **true** expression. The first part of the expression may be **false**, but as the second expression is **true** the entire expression would be **true**, regardless of the value of the variable *letter*.

4.2 THE 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.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

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
```

First, the condition is evaluated. If it is **true**, then the set of statements designated as "action1" is executed, and that is the end of the **if-else** statement. If, instead, the condition is **false**, the second set of statements designated as "action2" is executed, and that is the end of the **if-else** statement. The first set of statements ("action1") is called the action of the **if** clause; it is what will be executed if the expression is **true**. The second set of statements ("action2") is called the action of the **else** clause; it is what will be executed if the expression is **false**. One of these actions, and only one, will be executed — which one depends on the value of the condition.

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
```

PRACTICE 4.2

Write a script *printsindegorrad* that:

- will prompt the user for an angle
- will 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 (a 'd' means degrees, so **sind** is used; otherwise, for any other character the default of radians is assumed, so **sin** is used)
- will 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
```

cuu duong than cong . com

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
```

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
```

The **if-else** statement in this example chooses between two actions: printing an error message, or using the radius to calculate the area and then printing out the result. Note that the action of the **if** clause is a single statement, whereas the action of the **else** clause is a group of three statements.

4.3 NESTED IF-ELSE STATEMENTS

The **if-else** statement is used to choose between two actions. To choose from among more than two actions the **if-else** statements can be ***nested***, meaning one statement inside of another. For example, consider implementing the following continuous mathematical function y = f(x):

cuu duong than cong . com

```
y = 1   if x < -1
y = x^2 if -1 ≤ x ≤ 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, as follows:

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

Note that the && in the expression of the second **if** statement is necessary. Writing the expression as −1<=x<=2 would be incorrect; recall from Chapter 1 that that expression would always be **true**, regardless of the value of the variable x.

As the three possibilities are mutually exclusive, the value of y can be determined by using three separate **if** statements. However, this is not very efficient code: all three **logical** expressions must be evaluated, regardless of the range in which x falls. For example, if x is less than −1, the first expression is **true** and 1 would be assigned to y. However, the two expressions in the next two **if** statements are still evaluated. Instead of writing it this way, the statements can be nested so that the entire **if-else** statement ends when an expression is found to be **true**:

```
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
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

By using a nested **if-else** to choose from among the three possibilities, not all conditions must be tested as they were in the previous example. In this case, if x is less than -1, the statement to assign 1 to y is executed and the **if-else** statement is completed so no other conditions are tested. If, however, x is not less than -1, then the **else** clause is executed. If the **else** clause is executed, then we already know that x is greater than or equal to -1 so that part does not need to be tested.

Instead, there are only two remaining possibilities: either x is less than or equal to 2 or it is greater than 2. An **if-else** statement is used to choose between those two possibilities. So, the action of the **else** clause was another **if-else** statement. Although it is long, all of the above code is one **if-else** statement, a nested **if-else** statement. The actions are indented to show the structure of the statement. Nesting **if-else** statements in this way can be used to choose from among 3, 4, 5, 6, … the possibilities are practically endless!

This is actually an example of a particular kind of nested **if-else** called a ***cascading*** **if-else** statement. This is a type of nested **if-else** statement in which the conditions and actions cascade in a stair-like pattern.

Not all nested **if-else** statements are cascading. For example, consider the following (which assumes that a variable x has been initialized):

```
if x >= 0
   if x < 4
        disp('a')
   else
        disp('b')
   end
else
   disp('c')
end
```

4.3.1 The elseif Clause

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.

cuu duong than cong . com

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:

```
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**.

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
```

Note that in this example we only need one **end**. So, there are three ways of accomplishing the original task: using three separate **if** statements, using nested **if-else** statements, and using an **if** statement with **elseif** clauses, which is the simplest.

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
```

cuu duong than cong . com

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

    1.2100
```

QUICK QUESTION!

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 a 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.

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
```

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
```

CuuDuongThanCong.com https://fb.com/tailieudientucntt

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.

Another example demonstrates choosing from more than just a few options. The following function receives an integer quiz grade, which should be in the range from 0 to 10. The function then returns a corresponding letter grade, according to the following scheme: a 9 or 10 is an 'A', an 8 is a 'B', a 7 is a 'C', a 6 is a 'D', and anything below that is an 'F'. As the possibilities are mutually exclusive, we could implement the grading scheme using separate **if** statements. However, it is more efficient to have one **if-else** statement with multiple **elseif** clauses. Also, the function returns the letter 'X' if the quiz grade is not valid. The function assumes that the input is an integer.

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
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

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
```

In the part of this **if** statement that chooses the appropriate letter grade to return, all of the **logical** expressions are testing the value of the variable *quiz* to see if it is equal to several possible values, in sequence (first 9 or 10, then 8, then 7, etc.). This part can be replaced by a **switch** statement.

4.4 THE SWITCH STATEMENT

A **switch** statement can often be used in place of a nested **if-else** or an **if** statement with many **elseif** clauses. **Switch** statements are used when an expression is tested to see whether it is *equal to* one of several possible values.

The general form of the **switch** statement is:

```
switch switch_expression
  case caseexp1
    action1
  case caseexp2
    action2
  case caseexp3
    action3
  % etc: there can be many of these
  otherwise
    actionn
end
```

The **switch** statement starts with the reserved word **switch**, and ends with the reserved word **end**. The *switch_expression* is compared, in sequence, to the **case** expressions (*caseexp1*, *caseexp2*, etc.). If the value of the *switch_expression* matches *caseexp1*, for example, then *action1* is executed and the **switch** statement ends. If the value matches *caseexp3*, then *action3* is executed, and in general if the value matches *caseexpi* where *i* can be any integer from 1 to n, then *actioni* is executed. If the value of the *switch_expression* does not match any of the **case** expressions, the action after the word

cuu duong than cong . com

otherwise is executed (the *n*th action, *actionn*) if there is an **otherwise** (if not, no action is executed). It is not necessary to have an **otherwise** clause, although it is frequently useful. The *switch_expression* must be either a scalar or a string.

For the previous example, the **switch** statement can be used as follows:

switchletgrade.m

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

% First, error-check
if quiz < 0 || quiz > 10
    grade = 'X';
else
    % If here, it is valid so figure out the
    % corresponding letter grade using a switch
    switch quiz
        case 10
           grade = 'A';
        case 9
           grade = 'A';
        case 8
           grade = 'B';
        case 7
           grade = 'C';
        case 6
           grade = 'D';
        otherwise
           grade = 'F';
    end
end
end
```

Note

It is assumed that the user will enter an integer value. If the user does not, either an error message will be printed or an incorrect result will be returned. Methods for remedying this will be discussed in Chapter 5.

Here are two examples of calling this function:

```
>> quiz = 22;
>> lg = switchletgrade(quiz)
lg =
X

>> switchletgrade(9)
ans =
A
```

cuu duong than cong . com

As the same action of printing 'A' is desired for more than one grade, these can be combined as follows:

```
switch quiz
     case {10,9}
         grade = 'A';
     case 8
         grade = 'B';
       % etc.
```

The curly braces around the **case** expressions 10 and 9 are necessary.

In this example, we error-checked first using an **if-else** statement. Then, if the grade was in the valid range, a **switch** statement was used to find the corresponding letter grade.

Sometimes the **otherwise** clause is used for the error message rather than first using an **if-else** statement. For example, if the user is supposed to enter only a 1, 3, or 5, the script might be organized as follows:

switcherror.m

```
% Example of otherwise for error message
choice = input('Enter a 1, 3, or 5: ');

switch choice
    case 1
        disp('It''s a one!!')
    case 3
        disp('It''s a three!!')
    case 5
        disp('It''s a five!!')
    otherwise
        disp('Follow directions next time!!')
end
```

In this example, actions are taken if the user correctly enters one of the valid options. If the user does not, the **otherwise** clause handles printing an error message. Note the use of two single quotes within the string to print one quote.

```
>> switcherror
Enter a 1, 3, or 5: 4
Follow directions next time!!
```

Note that the order of the case expressions does not matter, except that this is the order in which they will be evaluated.

4.5 THE MENU FUNCTION

MATLAB has a built-in function called **menu** that will display a Figure Window with pushbuttons for the options. The first string passed to the **menu**

CuuDuongThanCong.com https://fb.com/tailieudientucntt

function is the heading (an instruction), and the rest are labels that appear on the pushbuttons. The function returns the number of the button that is pushed. For example,

```
>> mypick = menu('Pick a pizza','Cheese','Shroom','Sausage');
```

FIGURE 4.1 Menu figure window

will display the Figure Window seen in Figure 4.1 and store the result of the user's button push in the variable *mypick*.

There are three buttons, the equivalent values of which are 1, 2, and 3. For example, if the user pushes the "Sausage" button, *mypick* would have the value 3:

```
>> mypick
mypick =
     3
```

Note that the strings 'Cheese', 'Shroom', and 'Sausage' are just labels on the buttons. The actual value of the button push in this example would be 1, 2, or 3, so that is what would be stored in the variable *mypick*.

A script that uses this **menu** function would then use either an **if-else** statement or a **switch** statement to take an appropriate action based on the button pushed. For example, the following script simply prints which pizza to order, using a **switch** statement.

pickpizza.m

```
%This script asks the user for a type of pizza
% and prints which type to order using a switch

mypick = menu('Pick a pizza','Cheese','Shroom','Sausage');
switch mypick
    case 1
        disp('Order a cheese pizza')
    case 2
        disp('Order a mushroom pizza')
    case 3
        disp('Order a sausage pizza')
    otherwise
        disp('No pizza for us today')
end
```

This is an example of running this script and clicking on the "Sausage" button:

```
>> pickpizza
Order a sausage pizza
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

QUICK QUESTION!

How could the **otherwise** action get executed in this **switch** statement?

Answer

If the user clicks on the red "X" on the top of the menu box to close it instead of on one of the three buttons, the value returned from the **menu** function will be 0, which will cause the **otherwise** clause to be executed. This could also have been accomplished using a **case** 0 label instead of **otherwise**.

Instead of using a **switch** statement in this script, an alternative method would be to use an **if-else** statement with **elseif** clauses.

pickpizzaifelse.m

```
%This script asks the user for a type of pizza
% and prints which type to order using if-else
mypick = menu('Pick a pizza','Cheese', 'Shroom','Sausage');
if mypick == 1
    disp('Order a cheese pizza')
elseif mypick == 2
    disp('Order a mushroom pizza')
elseif mypick == 3
    disp('Order a sausage pizza')
else
    disp('No pizza for us today')
end
```

PRACTICE 4.5

Write a function that will receive one number as an input argument. It will use the **menu** function to display 'Choose a function' and will have buttons labeled 'fix', 'floor', and 'abs'. Using a **switch** statement, the function will then calculate and return the requested function (e.g., if 'abs' is chosen, the function will return the absolute value of the input argument). Choose a fourth function to return if the user clicks on the red 'X' instead of pushing a button.

4.6 THE "IS" FUNCTIONS IN MATLAB

There are a lot of functions that are built into MATLAB that test whether or not something is **true**; these functions have names that begin with the word "is". For example, we have already seen the use of the **isequal** function to compare arrays for equality. As another example, the function called **isletter** returns **logical** 1 if the character argument is a letter of the alphabet or 0 if it is not:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
>> isletter('h')
ans =
     1
>> isletter('4')
ans =
     0
```

The **isletter** function will return **logical true** or **false** so it can be used in a condition in an **if** statement. For example, here is code that would prompt the user for a character, and then print whether or not it is a letter:

```
mychar = input('Please enter a char: ','s');
if isletter(mychar)
    disp('Is a letter')
else
    disp('Not a letter')
end
```

When used in an **if** statement, it is not necessary to test the value to see whether the result from **isletter** is equal to 1 or 0; this is redundant. In other words, in the condition of the **if** statement,

```
isletter(mychar)
```

and

```
isletter(mychar) == 1
```

would produce the same results.

QUICK QUESTION!

How can we write our own function *myisletter* to accomplish the same result as **isletter**?

Answer

The function would compare the character's position within the character encoding.

myisletter.m

```
function outlog = myisletter(inchar)
% myisletter returns true if the input argument
% is a letter of the alphabet or false if not
% Format of call: myisletter(inputCharacter)
% Returns logical 1 or 0

outlog = inchar >= 'a' && inchar <= 'z' ...
         || inchar >= 'A' && inchar <= 'Z';
end
```

Note that it is necessary to check for both lowercase and uppercase letters.

cuu duong than cong . com

The function **isempty** returns **logical true** if a variable is empty, **logical false** if it has a value, or an error message if the variable does not exist. Therefore, it can be used to determine whether a variable has a value yet or not. For example,

```
>> clear
>> isempty(evec)
Undefined function or variable 'evec'.

>> evec = [];
>> isempty(evec)
ans =
     1

>> evec = [evec 5];
>> isempty(evec)
ans =
     0
```

The **isempty** function will also determine whether or not a string variable is empty. For example, this can be used to determine whether the user entered a string in an **input** function:

```
>> istr = input('Please enter a string: ','s');
Please enter a string:
>> isempty(istr)
ans =
     1
```

PRACTICE 4.6

Prompt the user for a string, and then print either the string that the user entered or an error message if the user did not enter anything.

The function **iskeyword** will determine whether or not a string is the name of a keyword in MATLAB, and therefore something that cannot be used as an identifier name. By itself (with no arguments), it will return the list of all keywords. Note that the names of functions like "sin" are not keywords, so their values can be overwritten if used as an identifier name.

```
>> iskeyword('sin')
ans =
     0

>> iskeyword('switch')
ans =
     1

>> iskeyword
ans =
    'break'
    'case'
    'catch'

    % etc.
```

cuu duong than cong . com

There are many other "is" functions; the complete list can be found in the Help browser.

Explore Other Interesting Features

- There are many other "is" functions. As more concepts are covered in the book, more and more of these functions will be introduced. Others you may want to explore now include **isvarname**, and functions that will tell you whether an argument is a particular type or not (**ischar**, **isfloat**, **isinteger**, **islogical**, **isnumeric**, **isstr**, **isreal**).
- There are "is" functions to determine the type of an array: **isvector**, **isrow**, **iscolumn.**
- The **try/catch** functions are a particular type of **if-else** used to find and avoid potential errors. They may be a bit complicated to understand at this point, but keep them in mind for the future! ■

Summary

Common Pitfalls

- Using = instead of == for equality in conditions.
- Putting a space in the keyword **elseif.**
- Not using quotes when comparing a string variable to a string, such as

```
letter == y
```

instead of

```
letter == 'y'
```

- Not spelling out an entire **logical** expression. An example is typing

```
radius || height <= 0
```

instead of

```
radius <= 0 || height <= 0
```

or typing

```
letter == 'y' || 'Y'
```

instead of

```
letter == 'y' || letter == 'Y'
```

Note that these are logically incorrect, but would not result in error messages. Note also that the expression "`letter == 'y' || 'Y'`" will *always* be **true**, regardless of the value of the variable *letter*, as `'Y'` is a nonzero value and therefore a **true** expression.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Writing conditions that are more complicated than necessary, such as

```
if (x < 5) == 1
```

instead of just

```
if (x < 5)
```

(The "==1" is redundant.)

- Using an **if** statement instead of an **if-else** statement for error-checking; for example,

```
if error occurs
    print error message
end
continue rest of code
```

instead of

```
if error occurs
    print error message
else
    continue rest of code
end
```

In the first example, the error message would be printed but then the program would continue anyway.

Programming Style Guidelines

- Use indentation to show the structure of a script or function. In particular, the actions in an **if** statement should be indented.
- When the **else** clause isn't needed, use an **if** statement rather than an **if-else** statement. The following is an example:

```
if unit == 'i'
    len = len * 2.54;
else
    len = len; % this does nothing so skip it!
end
```

Instead, just use:

```
if unit == 'i'
    len = len * 2.54;
end
```

- Do not put unnecessary conditions on **else** or **elseif** clauses. For example, the following prints one thing if the value of a variable *number* is equal to 5, and something else if it is not.

```
if number == 5
    disp('It is a 5')
elseif number ~= 5
    disp('It is not a 5')
end
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

The second condition, however, is not necessary. Either the value is 5 or not, so just the **else** would handle this:

```
if number == 5
    disp('It is a 5')
else
    disp('It is not a 5')
end
```

- When using the **menu** function, ensure that the program handles the situation when the user clicks on the red 'X' on the menu box rather than pushing one of the buttons. ■

MATLAB Reserved Words	
if	else
switch	elseif
case	otherwise

MATLAB Functions and Commands	
menu	isletter
isempty	iskeyword

Exercises

1. Write a script that tests whether the user can follow instructions. It prompts the user to enter an 'x'. If the user enters anything other than an 'x', it prints an error message; otherwise, the script does nothing.
2. Write a function *nexthour* that receives one integer argument, which is an hour of the day, and returns the next hour. This assumes a 12-hour clock; so, for example, the next hour after 12 would be 1. Here are two examples of calling this function.

```
>> fprintf('The next hour will be %d.\n',nexthour(3))
The next hour will be 4.
>> fprintf('The next hour will be %d.\n',nexthour(12))
The next hour will be 1.
```

3. Write a script to calculate the volume of a pyramid, which is 1/3 * base * height, where the base is length * width. Prompt the user to enter values for the length, width, and height, and then calculate the volume of the pyramid. When the user enters each value, he or she will then also be prompted for either 'i' for inches or 'c' for centimeters. (Note that 2.54 cm = 1 inch.) The script should print the volume

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt