Contents

CONDITIONAL STATEMENTS AND FLOW CONTROL

`if` Statements

Often we want to execute a command only if a certain test condition is satisfied. We use `if` statements to do this. There are a few different types of `if` statements.

Simple `if` statement

If you want to execute certain commands only when a certain test condition is met, use a simple `if` statement of the form:

```
if ( test condition )
     commands
end
```

- The "test condition" is any combination of relational and logical operators used to compare different variables/data.
  - You don't *need* the parentheses around the test condition, but it is a good idea to use parenthesis to improve code readability and avoid confusion and mistakes. Always make sure your parenthesis are balanced to prevent errors.
- The "commands" are any commands/statements/calculations that you have learned in MATLAB.
- An `end` is required for every opening `if`, which signifies the end of all commands that depend on the test condition. Anything outside of the `if` statement (before the opening `if` or after the closing `end`) is executed regardless of whether or not the test condition is satisfied – only the commands inside the `if` statement depend on the test condition.
- The indentation for the statements inside the `if` is not required but greatly improves the readability of presentation of the code. MATLAB will even automatically do indentation for you – simply select all the text in your M-File and right click for the "Smart Indent" option (CNTRL + I on Windows). Indentation is always recommended.

- Note that MATLAB is case sensitive – do not capitalize any letters in "`if`" or "`end`" as doing so will produce errors.

Let's jump right into an example to see how `if` statements work.

In `myfile.m`:

```
x = 5.4
y = 10
if (x>4)          % if the test condition is true, the following two lines will be executed
  y = y + 1;
  x = 2;          % if the test condition is false, the statements will not be executed
end
x
y
```

Sample output:

```
>> myfile

x =

    5.4000


y =

    10


x =

     2


y =

    11
```

In the example above, the test condition on the `if` statement was satisfied (it was true) since the value stored in `x` was indeed great than `4`. Because the test condition was true, the statements inside the `if` were executed, and the final values of `x` and `y` after the `if` statement were different than the initial values.

`if/else` statement

An `if/else` statement will allow you to execute certain commands if a test condition is true and execute other commands if the test condition is false. The commands inside the `else` are executed **only** if the test condition on the `if` statement is false. If the test condition on the `if` statement is true, the commands inside the `if` are executed, and the commands inside the `else` are skipped. Thus only the commands inside the `if` or the commands inside the `else` will be executed, never both. Let's look at an example of an `if/else` construct to illustrate.

In `myfile.m`:

```
x = 5
y = 10
if (x>y)             % Execute the next two lines if the test condition is true
    y = x;
    z = 100;
else
    z = 200;         % Execute this line if the test condition is false
end
x
y
z
```

Sample output:

```
>> myfile
x =

     5

y =

    10


x =

     5


y =

    10


z =

   200
```

As you can see, the values of `x` and `y` were unchanged and `z` was set equal to `200` because the test condition was false and the statement inside the `else` was executed.

`if/elseif` statement

If you have 2+ separate conditions that you need to check but only one can be satisfied at a time, you can use an `if/elseif` statement. In an `if/elseif` statement, a series of possible conditions are checked **in order**. The order is important because as soon as one condition is satisfied, the commands inside of that `if` (or `elseif)` are executed, and no other conditions are checked (i.e., all of the other `elseif` and an `else` if it exists are skipped).

In `myfile.m`:

```
x = 5;
if (x<0)                 % This is false, so the following commands are not executed
    z = 100;
    beep
elseif (x>=0 & x<10)% This is true, so the following commands are executed
    z = 200;             % and the if statement ends
    beep, beep
elseif (x<=10)           % This is true, but is not executed, since the previous was true
    z = 300;
    beep, beep, beep
end
z
```

Sample output:

```
> myfile
z = 200                  (with 2 beeps)
```

Note that once a test condition is found to be true, the statements inside that test condition are executed and the `if/elseif` statement is terminated even if later test conditions would be true too. This brings up a few important points to remember when using an `if/elseif` construct:

- Only one condition on an `if/elseif` can be true.
    - If you have 2 or more conditions that need to be checked and it is possible for both of them to be true simultaneously, you should use separate `if` statements rather than an `if/elseif`.
        - For example, this is required in homework 3, problem 4.
- When using an `if/elseif`, be careful with the order in which you enter your test conditions.
    - A practice problem that should be solved to illustrate this point is the following (solve this problem before proceeding with the notes):
        - Allow the user to enter their score on a 0-100 scale. Use an `if/elseif` statement to determine the user's equivalent letter grade with A being >=90, B being >=80, C being >=70, and D being >=60, and all other scores considered F.

        - Be careful! Putting test conditions in the wrong order will produce incorrect output for the letter grade. Be sure to test your code with an input in each of the grade ranges.
- Including an `else` on an `if/elseif` is optional.
    - An `else` is never required. Remember that it is only executed if all other test conditions are false.

Nested `if` statements

You can put `if` statements within `if` statements; this is called nesting and `if` statement. This is typically done if you only want to check a certain condition if another condition has already been satisfied. Thus, in this construct the nested `if` statement will be checked only when the `if` statement that it is inside of becomes true. Let's look at an example to illustrate.

In `myfile.m`:
```
score = input('Enter your score: ');
disp('Your grade is:')
if(score<=100 & score>=90)
    grade(1) = 'A';
    if(score>=90 & score<93)
        grade(2) = '-';
    elseif(score>=93 & score<97)
        grade(2) = ' ';
    elseif(score>=97 & score<=100)
        grade(2) = '+';
    end
    disp(grade)
else
    disp('B or worse')
end
```

Sample output:
```
>> myfile
Enter your score: 97.5
Your grade is:
A+
```

In this example, we only check the nested `if` statement if the score is within the range for an `A` grade. If the score is not within this range, the statements inside the `else` are executed and the nested `if` statement is skipped entirely.

We will execute the next example twice with different input values to illustrate the behavior of the nested `if` statement. In the second execution, the nested `if` statement will be skipped entirely because the `score` is not greater than `90`. Just remember in the nested `if` construct, the nested `if` statements are only checked when the outside `if` statement that they are inside of becomes true.

In `myfile.m`:

```
score = input('What was your test score?: ');
age = input('How old are you?: ');
if(score>90)
    if(age<=18)
        disp('You did very good... for a young person')
    elseif(age<=21 & age>18)
        disp('You did pretty good... for a young person')
    else
        disp('You are old')
    end
else
    disp('Your score is too low')
end
```

Sample output:

```
>> myfile
What was your test score?: 91
How old are you?: 14
You did very good... for a young person
>> myfile
What was your test score?: 81
How old are you?: 55
Your score is too low
```

While in MAE10 we will mostly be using `if` statements to compare alphanumeric values, do not underestimate the importance and power of `if` statement in programming. An `if` statement is the fundamental command that allows a computer to essentially make a decision – it is how logic is implemented into computer programs. You can think of an `if` statement as a condition put on an action.

For example, think about the anti-lock braking system (ABS) on your car. You car's computer uses what is essentially an `if` statement to determine when ABS needs to be engaged. If it detects that a wheel is locked up (based on some test conditions of wheel speeds, vehicle speed, etc), it engages the ABS system. If it does not detect that a wheel is locked up (the "`else`"), it does not engage the ABS system. As another example think about the icons on your computer's desktop – an `if` statement of sorts controls what to do when you click in a given area on your screen. For example, if you click on the MATLAB icon on your desktop, the computer runs MATLAB.exe and MATLAB opens. An `if` statement is again used to determine what command to run (e.g., MATLAB.exe) based on a test condition (where you clicked on the screen). Remember that your computer screen is essentially just an `(x,y)` grid with a size determined by your resolution.

switch and case

Although the functionality of switch and case is similar to that of if statements, the syntax used is quite different. A particularly useful application of switch is when comparing character strings. The general format of switch and case is the following.

```
switch variable
    case {options}
        statements
    case {options}
        statements
    …
    otherwise
        statements
end
```

- You must always begin with the keyword switch, followed by the variable you wish to compare against several options. This variable should already have a value stored in memory.
- Next, you list one or more different options. Before each option, you use the keyword case to signify that you want to compare the following option(s) to the original variable listed after the opening switch. For each option listed on a given case, MATLAB will determine if the variable listed after the switch is equal to that option. If the switch variable is equal to any of the options on a given case, the statements inside that case are executed and the entire switch/case is terminated.
- Similar to an if statement, only one case (or the otherwise) can be true. Once one is true and the statements inside are executed, the entire switch/case is terminated and the code proceeds after the end.
- An otherwise behaves exactly the same way as an else does for an if statement.

Let's look at an example using character strings:

In myfile.m:

```
animal = 'dog';
switch animal
    case {'cat'}
        disp('meow')
    case {'dog' , 'canine'}
        disp('woof')
    case {'sheep'}
        disp('baaaa')
    case {'duck'}
        disp('quack')
    otherwise
        disp('I have no idea')
end
```

Sample output:

```
>> myfile
woof
```

In the above example, the second `case` is satisfied and therefore the statements inside are executed. No other cases are true, and the `otherwise` is not used because the second `case` is true.

There are several important points to keep in mind when using a `switch/case` construct:

- The braces `{}` are **required** when you have more than one option on the same case (e.g., in the `{'dog', 'canine'}` example above).
    - The braces are optional with only a single option on one case, meaning that you do not *need* to put them, but it is safest to always do so if you are in doubt (similar to using brackets on the `disp()` command).
- Multiple options on the same case are separated by a comma. This is in contrast to `if` statements, which use the vertical bar | to indicate "or". In the above example, `{'dog','canine'}` means that the animal can be `'dog'` or `'canine'`, so the comma between the different options essentially means "or".
    - You cannot use any relational or logical operators on `switch/case` statements.
- In `switch/case`, you can only ask for equality between the variable (whatever is after the `switch` command) and the options on any given `case`. In others words, there is no way to ask if the variable is greater than, less than, etc, any of the options. For this reason, `switch/case` is typically used *only* for character strings.
    - Once again, you cannot use any relational or logical operators on `switch/case` statements.
- The `otherwise` on a `switch/case` functions the same way as an `else` on an `if` statement – the statements inside the `otherwise` are executed only if all other `case` options are checked and none of them are true.
    - An `otherwise` is optional and never required on a `switch/case`.
- You can include as many or as few `case` statements as you like on any given `switch/case` construct.
- Remember to include one `end` for every opening `switch` statement.
- It is okay to use other variables as options on a `case` statement – you do not need to put values or character strings directly as we did in the above example.
    - Be careful here! For example, including `dog` on a case (without the single quotes) will refer to a variable named `dog`. Including `'dog'` on a case will refer to the 3-letter word `dog`.