MATLAB Workshop 9 - Decision Making: **`if/elseif/else`**

Objectives: Learn about using **`if/elseif/else`** constructions to make decisions and take different actions based upon the decision.

MATLAB Features:

relational operators

Symbol	Meaning
==	is the same as
~=	is not the same as
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

logical operators

Symbol	Meaning
&	logical and
|	logical or
~	logical not

*one-sided **`if`** structure*

```
if (condition)
     % execute these commands when condition is true
          action 1
          action 2
          ...
end
```

*two-sided **`if/else`** structure*

```
if (condition)
     % execute these commands when condition is true
          action 1
          action 2
          ...
else  % default actions
     % execute these commands when condition is false
          action default1
          action default2
          ...
end
```

multiple selection ***if/elseif/else*** *structure*

```
if (condition_A)
     % execute these commands when condition_A is true
          action A1
          action A2
          ...
elseif (condition_B)
     % execute these commands when condition_B is true
          action B1
          action B2
          ...
elseif (condition_C)
     % execute these commands when condition_C is true
          action C1
          action C2
          ...
...
else  % default actions
     % execute these commands if none of the above are true
          action default1
          action default2
          ...
end
```

- **Decision making**

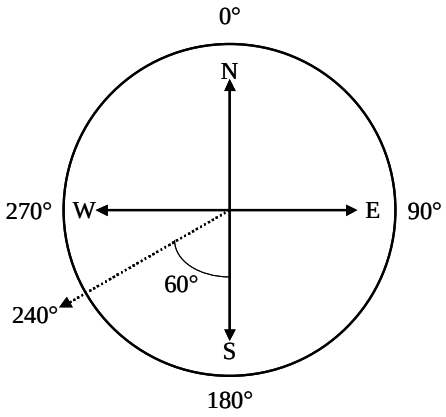
Thus far, we have been solving problems with a ***straight-line*** logic pattern. That is, we followed a sequence of steps (setting constant parameters, getting variable values, computations/calculations, and displaying results) that flowed directly from one step to another. The steps had to be in the programmed order and taken sequentially in order to achieve the desired result. Basically, we were using MATLAB to perform the same types of calculations that we could do by hand or with a calculator.

The real power of a computer (as opposed to a calculator) is the ability to make decisions (*decision making* or *branching actions*) and repeat the same set of steps over and over (*repetitive* or *looping actions*). This workshop will look at various aspects of decision making in MATLAB. Repetitive actions will be left until Workshop XX.

- **Civil Engineering problem**

As summer intern assigned to the surveying section of a civil engineering company, you have been asked to design a MATLAB function, named **`convert_heading`**, that will take a compass heading in degrees and return the proper bearing for later use.

The relationship between a compass **heading** in degrees and a compass **bearing** is illustrated at the right. A compass heading is a specification in degrees, between 0 and 359.999..., with N(orth) being 0 degrees. The arrow in the illustration shows a compass heading of 240 degrees.



A compass bearing is given in terms of **face** either north or south and then **turn** a specified number of **degrees** to the east or west. The compass bearing will look like

face (**degrees**) **turn**,

For the compass heading in the illustration, this becomes

south (60) west

which is read as face south, turn 60 degrees to the west.

The transformation table for converting compass headings into compass bearings is provided in the following table.

Heading (in degrees)	**Bearing Computation**
0 ≤ heading < 90	north (heading) east
90 ≤ heading < 180	south (180-heading) east
180 ≤ heading < 270	south (heading-180) west
270 ≤ heading < 360	north (360-heading) west

- **Designing the function**

The required function can be designed using the function design algorithm from Workshop 7.

`function` **`convert_heading`**

(1) Function to convert a compass heading in degrees to the equivalent bearing of face (degrees) turn.

(2) Produces **`face`**, **`degrees`**, **`turn`**.

(3) Needs **`heading`**.

(4) Algorithm

Need to make a decision based on the value of **`heading`**

if (0 ≤ **`heading`** < 90)

```
        bearing <-- north
        degrees <-- heading
        turn <-- east
elseif ( 90 ≤ heading < 180 )
        bearing <-- south
        degrees <-- 180 - heading
        turn <-- east
elseif ( 180 ≤ heading < 270 )
        bearing <-- south
        degrees <-- heading - 180
        turn <-- west
elseif  ( 270 ≤ heading < 360 )
        bearing <-- north
        degrees <-- 360 - heading
        turn <-- west
else  {Note: heading was not between 0&360 - Error!}
        bearing <-- Error!  Error!  Error!
        degrees <-- 0
        turn <-- Heading not in range 0 <= heading < 360
```

The function algorithm is a straightfoward applicaton of the decision table. An **if/elseif/else** construct is appropriate, rather than five separate **if** statements, for two reasons: (1) the decision for which branch to take depends only upon the value of **heading** and (2) the **if/elseif/else** construct will execute more rapidly than five separate **if** statements because the condition is only evaluated until a true branch is found in the **if/elseif/else** construct while all five separate **if** statement conditions must be evaluated. Thus, in the **if/elseif/else** construct, only three comparisons will be required on average while five comparisons will always be required when separate **if** statements are used.

Note that a default (**else**) clause was used to identify the possibility of an error. You should always ask yourself what can go wrong with a computation and make allowances to protect against the occurrence of an error. A common problem with computations is input values outside of an acceptable range.

- **Decisions, decisions, decisions**

The above design outline is making a decision about what actions to take based upon the value of **heading**. MATLAB provides two basic structures for making decisions and executing code based upon the decision. The more general structure is the **if/elseif/else** construct, which is described in this workshop. The **switch/case** construct is a more restricted, but useful, structure that is described in Workshop 10.

Making decisions in any programming language, MATLAB included, involves making a *logical comparison* between two or more values. Logical comparisons involve such statements as

is A the same as B?	**(A == B)**
is F greater than G?	**(F > G)**
is C less than or equal to D?	**(C <= D)**

The MATLAB code corresponding to the textual question is provided to the right of the query. These represent simple pairwise comparisons that use the MATLAB *relational operators* to ask a question.

If a comparison (*condition*) is true, then one or more actions associated with the statement need to be taken. If the statement (*condition*) is false, evaluation continues with the next statement (*condition*), skipping the actions associated with the false statement (*condition*).

More complex comparisons must be broken into pairwise comparisons using the *relational operators* which are then joined by a MATLAB *logical operator*. For example,

is (A the same as B) and (F less than G)?
((A == B) & (F < G))
is ((S less than or equal to H) and (H less than or equal to T))?
((S <= H) & (H <= T))
is ((A the same as B) and (F the same as G)) or (C less than D)?
(((A == B) & (F == G)) | (C <= D))

can also be evaluated as true or false. MATLAB (and other programming languages) can only perform pairwise comparisons. More on design and interpretation of logical comparisons can be found in XXXX.

(1) Using **if/elseif/else** to make decisions.

MATLAB uses the **if/elseif/else** construct to execute different sets of actions depending upon the truth of a condition. The basic form of the construct is

```
if (condition_A)
    % execute these commands when condition_A is true
        action A1
        action A2
        ...
elseif (condition_B)
    % execute these commands when condition_B is true
        action B1
        action B2
        ...
elseif (condition_C)
    % execute these commands when condition_C is true
        action C1
        action C2
        ...
...
else  % default actions
    % execute these commands if none of the above are true
        action default1
        action default2
        ...
end
```

Note that **if**, **elseif**, **else**, and **end** are all blue (both in the Command Window and the Editor). This indicates that these words are *reserved keywords* in MATLAB that cannot be used as a variable or function name. Their meanings are fixed and cannot be changed. Also note that the actions associated with any of the keywords are indented under the keyword. The MATLAB Editor will automatically indent for you. The purpose of indentation is to easily identify which actions belong with which keyword and condition. Finally, the actions can be any legitimate MATLAB command, including, but not limited to, assignment statements, function calls (such as **disp** and **input**), another **if/elseif/else** construct, or a looping construct.

Two abbreviated forms of the general construct are frequently used. The first is the simple **if** construct

```
if (condition)
    % execute these commands when condition is true
        action 1
        action 2
        ...
end
```

and the second is the **if/else** construct

```
if (condition)
    % execute these commands when condition is true
        action 1
        action 2
        ...
else  % default actions
    % execute these commands when condition is false
        action default1
        action default2
        ...
end
```

The **if** construct is frequently referred to as a *one-sided decision* - actions are only taken if the condition is true. The **if/else** construct is frequently referred to as a *two-sided decision* - actions are taken regardless of whether the condition is true. When the condition is true, the "true" actions are executed. When the condition is false, the default actions are executed.

If you are working from a reasonably well-developed outline, such as above, coding a decision in MATLAB is relatively straight-forward. If you are not working from such an outline, good luck!

For example, the algorithm outline in the function design can be (partially) encoded as

```
% convert heading to bearing
    if ((0 <= heading) & (heading < 90) )  % first quadrant
        face = 'north';
        degrees = heading;
        turn = 'east';
    elseif ( ?? )  % second quadrant
        ...
    elseif ( ?? )  % third quadrant
        ...
    elseif ( ?? )  % fourth quadrant
        ...
    else  % default - heading not valid
        ...
    end
```

Create the function **convert_heading** in the MATLAB editor. Be sure to follow the format

```
function [output list] = fcn_name(input list)
% function description
% header information
```

```
% variable dictionary

% algorithm
```

Save your function in your current directory. Test your function. How many tests are required to ensure that the function is operating properly?

- **Testing your script**

You have just spent a modicum of effort in creating a function. It now needs to be tested to in order to assure that it provides the appropriate responses for any given input.

You should always test your functions with values for which you know the answer!! If it works for those problems, you can have some confidence that it will work for other values for which you do not know the answer.

So, test your function. How many different tests are required to ensure that the function is operating properly?

- **Exercises**:

1. Calculation of energy loss due to fluid flow through a pipe is common to several engineering disciplines. Energy loss can be calculated by

$$h_L = f\left(\frac{L}{D}\right)\left(\frac{V^2}{2}\right); \quad V = \frac{Q}{A}; \quad A = \frac{\pi D^2}{4}; \quad Re = \frac{DV\rho}{\mu}$$

where:
h_L : energy loss per mass of fluid flowing, (J/kg)
f : friction factor, dimensionless
L : pipe length, m
D : pipe diameter, m
V : average fluid velocity, m/s
Q : volumetric flow rate, m^3/s
A : pipe cross-sectional area, m^2
Re : Reynolds number, dimensionless
ρ : fluid density, kg/m^3
μ : fluid viscosity, kg/(ms)

The friction factor, f, is calculated as

$$f = 64/\mathrm{Re} \quad when \quad Re \le 2{,}000$$

and

$$f = \left\{(-2.01)\ln\left[\frac{-5.0452}{Re}\ln\left(\frac{5.8506}{Re^{0.8981}}\right)\right]\right\}^{-2} \quad when \quad Re > 2{,}000$$

You have been asked to design a MATLAB function that will calculate the energy loss per mass of flowing fluid for fluid flow in a pipe given the pipe diameter, pipe length, fluid volumetric flow rate, fluid density, and fluid viscosity (all in SI units).

2. Given a year between 1982 and 2048, inclusive, the date for any Easter Sunday can be computed from the set of relations

   A = Year modulus 19
   B = Year modulus 4
   C = Year modulus 7
   D = (19*A + 24) modulus 30
   E = (2*B + 4*C + 6*D + 5) modulus 7

   Easter Sundy is then Sunday, March (22 + D + E). Note that (22+D+E) could be greater than 30, giving a date that is really in April!

   Design a MATLAB function that when given the year, returns the day of the week on which Easter Sunday falls, the month in which Easter Sunday falls, and the date of the month on which Easter Sunday falls. If the year lies outside the permissible range, the day of the week should return the phrase "Error!!!", the month should return a phrase indicating that the year is not in the acceptable range, and the date of the month should return the offending year.

3. Design a function that, given an (x,y) data point, will respond with a text message locating the point on the cartesian coordinate graph. Possible responses include: Quadrant 1, Quadrant 2, Quadrant 3, Quadrant 4, Positive x-axis, Negative x-axis, Positive Y-axis, Negative Y-Axis, and Origin. Note: efficient algorithm construction for this function will require that you "nest" **`if/else`** statments within **`if/else`** statements.

Recap: You should have learned

- The relational operators in MATLAB.
- The logical operators in MATLAB.
- The utility of outlining a script before coding.
- The one-sided **`if`** structure.
- The two-sided **`if/else`** structure.
- The multiple condition **`if/elseif/else`** structure.
- The multiple condition **`switch/case`** structure.
- That the **`switch/case`** structure should only be used for integer or character **`switch`** values.
- To test your scripts with known problems before trusting them with unknown problems.