Logical Operations and Decision Branching Assignment

Problem 1

Given *v*=[4 -1 2 3 1 -2 5 0] and *u*=[5 -1 0 3 -3 2 1 5]. Evaluate the follwing expression without using MATLAB. Then, check the answers with MATLAB.

1. ~ ~u
2. v == ~u
3. u == abs(*v)*
4. *v* >= *u+v*

```
% define vectors
v = [4, -1, 2, 3, 1, -2, 5, 0];
u = [5, -1, 0, 3, -3, 2, 1, 5];

% check expressions
~~u
```

```
ans = 1×8 logical array
   1   1   0   1   1   1   1   1
```

```
v == ~u
```

```
ans = 1×8 logical array
   0   0   0   0   0   0   0   1
```

```
u == abs(v)
```

```
ans = 1×8 logical array
   0   0   0   1   0   1   0   0
```

```
v >= u+v
```

```
ans = 1×8 logical array
   0   1   1   0   1   0   0   0
```

Problem 2

Write code to deteremine the real roots of a quadratic equation $ax^2 + bx + c = 0$. The user should define the values for *a, b,* and *c* at the top of the code. To calculate the root, the code should first calculate the discriminant *D*, given by:

$$D = b^2 - 4ac$$

- If *D>0* the program should use the `disp` function to print "The equation has two roots," and the roots should be displayed on the next line.
- If *D=0,* the program should display the message "The equation has one root," and the root should be displayed on the next line.
- If *D<0,* the program should display the message "The equation has no real roots."

Run the code on the following test cases:

1. $3x^2 + 6x + 3 = 0$
2. $-3x^2 + 4x - 6 = 0$
3. $-3x^2 + 7x + 5 = 0$

```
% define coefficients of 2-polynomial
a = [3, -3, -3];
b = [6, 4, 7];
c = [3, -6, 5];

% calculate discriminant
D = b.^2 - 4 .* a .* c;

% iterate over the disciminants to find roots for each polynomial
for i = 1:length(D)
        % evaluate the type of root(s)
        if D(i) > 0
                % two real roots
                root1 = ( -b(i) + sqrt(D(i)) ) / ( 2*a(i) );
```

```
                root2 = ( -b(i) - sqrt(D(i)) ) / ( 2*a(i) );

                % display roots
                % disp(sprintf(X)) = fprintf(X, \n)
                fprintf("Eq %d. The equation has two real roots:\n\t%f\n\t%f\n", i, root1, root2)
        elseif D(i) == 0
                % one real root
                root1 = -b(i) / ( 2*a(i) );

                % display root
                fprintf("Eq %d. The equation has one real root:\n\t%f\n", i, root1)
        elseif D(i) < 0
                % imaginary roots
                % display text
                fprintf("Eq %d. The equation has no real roots.", i)
        end
end
```

```
Eq 1. The equation has one real root:
        -1.000000
Eq 2. The equation has no real roots.
Eq 3. The equation has two real roots:
        -0.573384
        2.906718
```

Optional Advanced Exercise:

Exercise 1: Make a unit conversion live script

Write a live script that converts pressure units. The user will be asked to give units in either Pa and psi, or atm and the program will give the equivalent value in another unit system specified by the user. The program should ask the user to input (1) the amount, (2) the current unit (compare string data using `strcomp`), and (3) the desired unit. If you want, you can use the live script's dropdown feature under Control in the Toolbar.

Use the program to:

(a) Convert 70 psi to Pa

70 psi ~= 482625 Pa

(b) Convert 800 Pa to psi

800 Pa ~= 0.116032 psi

(c) Convert 1 atm to psi

1 atm ~= 14.696 psi

This does not use if statements but instead uses matrix transforms (arguably cooler). The input is a scaled unit vector in a Pa psi atm space. This is then multiplied by a matrix such that the output is a vector containing the transforms from each basis unit to another. If the user gives [1, 0, 0] that is 1 Pa and the output will be [Pa to Pa, Pa to Psi, Pa to Atm] or [1, 1.45e-4, 9.87e-6]. The user also supplies a target unit selection which just picks out a single element from the output vector.

```
% input a value with units
pressure = 1;
units = 3;

% designate target units
target_units = 2;

% define a vector for unit strings
u_strs = ["Pa", "psi", "atm"];

% define input vector
user_in = zeros(3,1);
user_in(units) = pressure;

% define transformation matrix
pa_to_psi = 1.4504e-4;
pa_to_atm = 9.87e-6;
psi_to_atm = 6.804596e-2;
pressure_transform = [1, pa_to_psi.^-1, pa_to_atm.^-1; pa_to_psi, 1, psi_to_atm.^-1; pa_to_atm, psi_to_atm, 1];

% multiply user input by transform
user_out = pressure_transform * user_in;
```

```
% display
fprintf("%f %s = %f %s\n", pressure, u_strs(units), user_out(target_units), u_strs(target_units))
```

```
1.000000 atm = 14.695950 psi
```