EGR 111
Conditional Execution

This lab shows how to write a program that performs differently depending on a given condition or user input. The lab also explains logical operators and functions.

New MATLAB Commands: `&, |, ~, any, all, sum, if, else, elseif, end, input, disp`

1. Logical Operators

Suppose you wanted to see if the variable x is within a given range, say between 1 and 10 (that is to say that $x \geq 1$ and $x \leq 10$). For this type of operation, we need the following logical operators.

Table 1. Logical Operators

Operator	Description
`&&`	AND. The expression x && y is true if x is true **and** y is true. The variables x and y must be scalar values.
`||`	OR. The expression x || y is true if x is true **or** y is true (or both). The variables x and y must be scalar values.
`~`	NOT. The expression ~x is true if x is **not** true.

The AND operator is 1 if the first operand is *true* AND the second operand is *true* (that is both operands are 1).

```
>> 1 && 1
ans =
     1
```

The OR operator is 1 if the first operand is *true* OR the second operand is *true* (that is if either operand is 1 or both operands are 1).

```
>> 1 || 0
ans =
     1
```

The NOT operator is *true* if the operand is NOT *true* (that is if the operand is 0).

```
>> ~0
ans =
     1
```

Try out the other inputs to the AND, OR, and NOT operators to verify that they work the way you think they do.

To test to see if x is between 1 and 10 (that is to say that $x \geq 1$ and $x \leq 10$), we would write the following (choosing x = 4 as an arbitrary example value):

```
>> x = 4
x =
     4
>> x >= 1 && x <= 10
ans =
  logical
   1
```

Try out other values for x to see if they work as expected.

A common error: You might think that you could test if x is between 1 and 10 using an expression like "`1 <= x <= 10`", but this doesn't work in MATLAB. For example, if `x = -4`, MATLAB would first compute "`1 <= x`" which would be 0 (false), and then MATLAB would compute "`0 <= 10`" which is 1 (true). The expression "`1 <= x <= 10`" turns out to be true for ALL values of x, which is not what we want, so use "`x >= 1 && x <= 10`" instead.

Just like with relational operators from the previous experiment (>, <, >=, <=, ==, and ~=), one must be careful with operator precedence. In the above command, the relational operators (>= and <=) are computed first, then followed by the logical operator (AND) (see Table 2).

Table 2. Operator Precedence

Precedence	Operation
1	Parentheses
2	Exponentiation (^)
3	Logical NOT (~)
4	Multiplication (*) and division (\ )
5	Addition (+) and subtraction (-)
6	Relational operators (> < >= <= == ~=)
7	Logical AND (&&)
8	Logical OR (||)

If two operators have the same precedence, the expression is executed left to right.

Exercise 1: Evaluate each of the following expressions by hand first and then use MATLAB to check your answer.

```
>> x = -2; y = 5;
>> -5 < x & x < -1
  ans = _____

>> ~(y < 7)
  ans = _____

>> ~y < 7
  ans = _____

>> ~(y >= 8) | (x < -1)
  ans = _____
```

2. Logical Functions

MATLAB has other built-in functions that can aid in determining logical relationships among values (see Table 3). Try the following functions on some sample inputs to see how they work.

Table 3. Helpful Functions

Function	Description
`any(A)`	If A is a vector, returns 1 if any of the elements A is *true* (non-zero) and 0 otherwise.
`all(A)`	If A is a vector, returns 1 if all of the elements in A are *true* (non-zero) and 0 otherwise.
`find(A)`	If A is a vector, returns the indices of the elements that are *true* (non-zero).
`sum(A)`	If A is a vector, returns the sum of the values in the vector. If A is a matrix, sum(A) returns a row vector where each element is the sum of the columns.

For example, in order to count the number of negative elements in the vector x, we could use the following commands:

```
>> x = [-20 -10 10 20 30]
x =
   -20   -10    10    20    30
>> sum(x < 0)
ans =
     2
```

The `x < 0` command above results in the vector [1 1 0 0 0] because the comparison x < 0 is true for the first two elements. Then the command `sum(x < 0)` adds up the elements in the vector: 1 + 1 + 0 + 0 + 0 = 2.

Suppose we wanted to count the number of elements in matrix A that are less than zero. The command "sum(A < 0)" will add up the columns of the matrix, resulting in a row vector as shown below. The command "sum(sum(A < 0))" will first add up the columns and then add up the total for each column, resulting in the total number of elements that are less than zero.

```
>> A = [-10 -20; 10 20]
A =
   -10   -20
    10    20

>> A < 0
ans =
  2×2 logical array
   1   1
   0   0

>> sum(A < 0)
ans =
     1     1

>> sum(sum(A < 0))
ans =
     2
```

Exercise 2: Write a function called numZeros that has one input argument and one output argument. The input will be a vector, and the output is the number of elements in the input argument that are equal to 0. For example, if the input is x = [-1, -2, 0, 1, 2], then the output is 1 because the input contains one zero. If the input is x = [0, 1, 0, 2], then the output is 2.

Checkpoint 1: Show the instructor your function from Exercise 2.

3. User Input

Sometimes a program needs to prompt the user for information. Consider the following command.

```
>> x = input('Enter a number: ')
Enter a number: 5
x =
     5
```

The input to the `input` command is the string `'Enter a number: '`. Recall that in MATLAB strings are enclosed in single quotes. When MATLAB executes the `input` command, it prints the string to the Command Window and waits for the user to type a value and hit the Enter key. In the above example, the user typed "5" and then hit Enter. MATLAB then placed the supplied value into the variable x. Since there is no semicolon after the `input` command to suppress printing, MATLAB also printed the value of x to the command window.

4. Printing the Value of Variables

If we want to print the value of a variable, we can simply type it into the Command Window:

```
x = 5;
x
```

Then MATLAB will print the following:

```
x =
     5
```

The `disp` function is similar to typing a variable name in that it displays the value of the variable, but it does not print the name of the variable. For example, suppose we type the following commands:

```
x = 5;
disp(x)
```

Then MATLAB will print the following:

```
     5
```

We also can use the `disp` function to print text to the Command Window:

```
disp('Hello!')
```

Then MATLAB will print the following:

```
Hello!
```

Characters enclosed in single quotes are called character arrays. MATLAB also supports strings, which are characters enclosed in double quotes (for example "This is a string!"). The `disp`

function can print both character arrays and strings, but in this course we will only use character arrays.

MATLAB also has a function called `fprintf` that gives you much more control over how the values are printed. We won't need the `fprintf` function in this course, but if you want to see how it works, read the help file for `fprintf`.

5. Conditional Statements

Thus far in this course, we have only seen MATLAB scripts that execute the same commands every time they are run. However, many programs need to execute a set of commands only if a given condition exists, or need to execute one set of commands if a given condition exists and another set of commands if the condition does not exist. There are three structures to handle this situation: if-end, if-else-end, and if-elseif-else-end. These structures are discussed in turn.

5.1 The if-end structure

Let's write a script file that selects a number between 1 and 10 and prompts the user to guess the number. If the user gets the number correct, the program should print a congratulatory message. Type the following commands into a new script file.

```
clear
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('You guessed it!')
end
```

In the script file above, the line "`a = 4;`" selects a number (any number could have been selected). Then the `input` command prompts the user for a value and places the value that the user types into the variable x. The `if` statement evaluates the relational operator `x == a`. If the comparison `x == a` is true, then the `disp` command that comes after the `if` statement is executed. If the user guesses the wrong number, `x == a` is false, and nothing is printed.

There can be any number of commands in the `if` statement as shown below.

```
clear
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('Congratulations!')
    disp('You guessed it!')
    disp('Yea!')
end
disp('That was fun.')
```

The `end` statement marks the end of the commands that are executed if the comparison `x == a` is true. Also, note that it is good programming practice and strongly recommended to indent the commands that are inside the `if-end` statement to make the program easier for humans to read, but MATLAB does not care if the commands are indented or not.

MATLAB requires an "`end`" for every "`if`". The final `disp` statement above follows the "`end`", and the command window will print "That was fun." regardless of whether or not x equals a.

Run the above program a few times to verify the operation of the script.

A Common Error: Usually when we use the `if` statement, the comparison (such as "x == 4") results in a scalar value (which is a single number, not a vector or matrix). If the comparison in the "if" statement results in a vector or matrix, the command is executed if **ALL** of the elements are true. Since this behavior can lead to confusion, **it is strongly recommended to only use a comparison that results in a scalar in an `if` statement**.

5.2 The if-else-end structure

In the previous example, if the user guessed incorrectly, the program printed "That was fun.", which might be confusing to the user. So let's change the program so that it tells the user if they guessed wrong.

```
clear
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('You guessed it!')
else
    disp('Nope.')
end
disp('That was fun.')
```

In the script above, the command `disp('You guessed it!')` is executed if `x == a` is true, and the other command `disp('Nope.')` is executed if `x == a` is false. The if-else-end structure is used when one set of commands needs to be executed when a condition is true, and a different set of commands needs to be executed when the condition is false. Again, note that regardless if there is an `else` command, every `if` requires an `end`.

Run the above script a few time to verify how it works.

5.3 The if-elseif-else-end structure

Next, let's tell the user if he or she is close to the correct number. If the user's guess is close to the correct number, then the difference a-x will be close to zero. For example, if a = 4 and x = 5, then a-x = 4-5 = -1. If a = 4 and x = 3, then a-x = 4-3= 1. So if the absolute value of a-x is small, let's say less than or equal to 1, then the guess is close (see below).

```
clear
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('You guessed it!')
elseif abs(a-x) <= 1
    disp('You are very close.')
else
    disp('Nope.')
end
disp('That was fun.')
```

In the script above, if `x == a` is true, the string `'You guessed it!'` will be printed. Otherwise, if `x == a` is false, then the comparison `abs(a-x) <= 1` is computed, and if it is true, then MATAB will print the string `'You are very close.'`. In particular, if the user guessed 3 or 5, then `abs(a-x)` will be 1, so the string `'You are very close.'` will be printed. If `x == a` is false, and `abs(a-x) <= 1` is false, then `'Nope.'` will be printed.

Run the script above a few times to verify that it works as expected.

In the if-elseif-else-end structure, there can be multiple "elseif" statements. Regardless of how many "elseif" statements there are, there needs to be just one "end" statement. The "else" statement is optional, and there can be at most one "else" statement.

Exercise 3: Modify the script above so that it prints "You are close, but not that close" if the value of abs(a-x) equals 2 (that is if the user guesses 2 or 6).

Exercise 4: Modify the script above so that it prints 'Too high', 'Too low', or 'Just right' if the users guess is too high, too low, or correct respectively. Please name the script file ConditionalEx4.m.

Checkpoint 2: Show the instructor your script file for Exercise 4.