Contents

RELATIONAL AND LOGICAL OPERATORS 1
Relational Operators 1
Logical Operators 1
Using Relational and Logical Operators with Scalars 2
Using Relational Operators with Numerical Arrays 5
Using Relational Operators with Character Arrays 6

RELATIONAL AND LOGICAL OPERATORS

Relational Operators

Often we want to compare values of different variables in order to determine what to do next. As you might have guessed from the name, relational operators are used to determine the relationship between different variables. There are many relational operators that allow us to compare values. You should be familiar with many of these from mathematics, however, pay careful attention to how they are represented and used in MATLAB.

`>`	greater than
`<`	less than
`==`	equal to (Notice the *double* equal sign. One equal sign is the assignment operator.)
`>=`	greater than or equal to
`<=`	less than or equal to
`~=`	not equal to

When using a relational operator to compare alphanumeric data (letters and numbers), the result will be a logical variable. Logical variables are a different type of variable and only have two possible values: `1` (true) or `0` (false). To compare logical data, we must use logical operators, which are discussed in the next section.

Logical Operators

The relational operators above allow us to compare alphanumeric data (e.g., letters and numbers). However, if we want to compare logical data (e.g., true or false, represented by `1` or `0` in MATLAB), we need to use logical operators. The syntax for using logical operators in MATLAB is the following:

`&`	and	(both statements must be true)
`|`	or	(either or both statements can be true)
`~`	not	(the opposite of the statement)
`xor()`	exclusive or	(only one statement can be true)

In most cases, logical operators are used together with relational operators. In this context, logical operators are used between relational statements (the output of each being a `1` or `0`), resulting in an overall output of just a single `1` or `0`, which illustrates whether the overall statement is true or false.

Using Relational and Logical Operators with Scalars

Let's begin by going through some examples using relational operators.

```
>> x  = 5.4

x =

    5.4000

>> y = 3.9

y =

    3.9000

>> x == y

ans =

     0
```

(`0` means false in MATLAB)

```
>> x > y

ans =

     1
```

(`1` means true in MATLAB)

```
>> x >= y

ans =

     1

>> x ~= y

ans =

     1
```

```
>> x > y/50

ans =

     1
```

When we compare two pieces of alphanumeric data, MATLAB returns `0` (false) or `1` (true). Often we want to do some action if two or more conditions are met. For example, if you have done your homework and it is earlier than midnight, you will eat a snack before bed. This will require the use of logical operators. Logical operators, as you can infer from the name, are used on logical variables (e.g., variables that have a value of either `0` or `1` and are of the logical type). Here are the rules for using logical operators:

```
True and True   → True          1 & 1 → 1    (are both statements true?  true!)
True and False  → False         1 & 0 → 0    (are both statements true?  false!)
False and True  → False         0 & 1 → 0
False and False → False         0 & 0 → 0

True or True   → True           1 | 1 → 1    (is either statement true? true!)
True or False  → True           1 | 0 → 1    (is either statement true? true!)
False or True  → True           0 | 1 → 1
False or False → False          0 | 0 → 0

not True  → False                 ~1 → 0     (the opposite of true? false!)
not False → True                  ~0 → 1

exclusive True or True   → False   xor(1,1) → 0  (is only 1 statement true? false!)
exclusive True or False  → True    xor(1,0) → 1  (is only 1 statement true? true!)
exclusive False or True  → True    xor(0,1) → 1
exclusive False or False → False   xor(0,0) → 0
```

Next, let's go through some examples using logical operators together with relational operators:

```
>> 7 == 7 & 6 > 7
```
(true and false → false)

```
ans =

     0
```

```
>> 7 == 7 & 6 < 7
```
(true and true → true)

```
ans =

     1
```

```
>> 7 == 7 | 6 > 7
```
(true or false → true)

```
ans =

     1
```

```
>> 7 == 9 | 10 > 11
```
(false or false → false)

```
ans =

     0
```

```
>> ~(7 == 9)
```
(not (false) → true)

```
ans =

     1
```

```
>> ~(7 == 7)
```
(not (true) → false)

```
ans =

     0
```

Note that the way we write a sequence of relational operators may be different than what you are used to from mathematics. For example, when using relational operators "on paper" it is okay to write something along the lines of `(1 < x < 5)` to indicate that `x` is strictly greater than `1` and strictly less than `5`. However, this syntax is **not** allowed in MATLAB. To write the same statement in MATLAB, it would have the form `(1 < x & x < 5)`. **In other words, we must use only one relational operator at a time, and link several of them together using logical operators.**

One other important point about using the relational operator for equality (==) involves numerical/rounding errors. The short answer is that you should **never** ask for equality between two variables unless they are integers, character strings, or logical variables. Not only will this be important in MAE10 homework problems, but it is inherent to all computational programming. Please see the document on numerical and rounding errors for more details and additional explanation.

Here are some additional examples using variables. It is a good idea to use parentheses to avoid confusion, group terms, and to control the order of execution.

```
>> x = 100; y = 8.1;
>> (x > y) | (x == 0)

ans =

     1

>> ( (y*3)<(x*4) ) & (x>5)

ans =

  1
```

Using Relational Operators with Numerical Arrays

You can also compare entire arrays of data using the same relational operators. For example:

```
>> quiz1 = [99,88,77];
>> quiz2 = [90,90,89];
>> quiz1 < quiz2

ans =

     0     1     1
```

Note that in order to compare two arrays, they must be the same size. An array of true/false values (`1`s and `0`s) is returned, which is the same size as the two arrays compared. Thus, you can see that using relational operators between two arrays does not determine ***if*** the two arrays are equal, it determines ***where*** the two arrays are equal – it compares every element of one array to every element of the other array. The true/false values that result can be stored in a logical array. A logical array is an array that contains only true/false values. It is a different class of variable (logical) and consumes less computer memory than floating point numbers or character strings because each element only has two possible values (`1` or `0`).

```
>> a = quiz1 < quiz2

a =

     0     1     1

>> whos
  Name        Size              Bytes  Class      Attributes

  a           1x3                   3  logical
```

You can also compare all individual elements in an array to a single value (a scalar). The output will automatically be a logical array of the same size as the input array.

```
>> quiz1 > 79

ans =

     1     1     0
```

Using Relational Operators with Character Arrays

Although it is uncommon, relational operators can also be used to compare character data. In general, comparison between character arrays is handled by the `switch` function in MATLAB, which will be covered later. The only relational operator that should be used with character data is the double equals (==) to ask for equality. The important point to remember when asking for equality between character data is that MATLAB is case sensitive in this regard. The following example illustrates this point:

```
>> 'a' == 'a'

ans =

   1

>> 'a' == 'A'

ans =

   0

>> 'name' == 'name'

ans =

   1   1   1   1

>> 'name' == 'namE'

ans =

   1   1   1   0
```

Finally, notice that when we compare character arrays using a relational operator, the output is an array of the same size. Comparing character arrays follows the same rules as comparing numerical arrays as discussed previously.