Lesson 1, Week 3:
(comparison operators, logical operators)

AIM

—

After this lesson, you will be able to

* Name and type the five comparison operators used on this course
* Name and type the three logical operators used on this course
* Discuss which kinds of values can be compared with the comparison operators
* Write long logical tests which combine names with comparison as well as logical operators

Looking back, looking forward

This week, we study things that are often done much earlier in a programming course. But remember our slogan "Small steps, leave nothing out, everything makes sense". What you have learnt so far is about creating and modifying values, often values that are stored by binding them to the names of variables.

The code that created these values consisted of a sequence of valid expressions, carried out in the order they appear in the code. We will call such a sequence a *code path*. Let's recap what you've learnt about creating code paths. We'll use as headings the four components of valid Julia code: values, names, operators and delimiters.

Values String values (of type `String`) which consist of characters (of type `Char`), number values (types `Int64` and `Float64`) and the logical constants `true` and `false` (the possible values of type `Bool`).

Names These include the valid variable names, which are also valid names for functions, as well as the reserved keywords which cannot be used for names[1] and the names in Julia's type system.

[1] So far you've seen that the reserved keywords include `false`, `true`, `for`, `function`, `return`, `end`, `global` and `local`. We'll do just a few more, so about half of the reserved keywords in Julia are not part of this course.

Operators So far, you've seen the assignment operator `=`, the string operators `*` and `^`, the number operators `+`, `-`, `*`, `\` and `^`, the `in` operator used to make comprehensions, and the type operator `::`.

Delimiters You've seen `" "` (for strings), `' '` (for characters), `[ ]` (for arrays and for indexing into arrays and strings), `( )` (for a function's argument, and also to specify the order of arithmetic operations), and `,` (as separators for the elements of an array and for the variables in an argument list).

What you haven't seen much of is the applied formal logic that I spoke about such a lot in week 1. It is time to remedy that. Recall that formal logic is based on the two logical constants `true` and `false`[2]. In what follows, we look first at comparison operators, which generate those values, and then at the logical operators `!`, `&&` and `||`.

This will enable you create logical tests, which Julia uses to create branching code[3], in which the calculation could proceed in one of two ways, and the way chosen depends on whether the result of a logical test is true or false.

Comparison operators

In this course, we use the following comparson operators[4]:

Symbol	Result
`==`	equality test
`>`	true if left hand side greater than right
`<`	true if left hand side smaller than right
`>=`	ditto for greater than or equal to
`<=`	ditto for less than or equal to

These are, of course, taken from standard mathematics, with the exception of the equality test. In Julia, as in all computer languaes, a strict separation is made between assignment and equality testing. Since assignment already uses a single equals sign, a different symbol is needed for equality testing, and Julia's designers chose the double equals sign.

All of these operators are written as they would be in maths: between a left hand side value and a right side value. The result of such a comparison is always one of the constants `true` or `false`. Let's see this in action.

[DEMO: examples of comparison tests using number, character and string values.]

[2] It is true that some formal logic use a third and even a fourth constant, and to a very limited extent that also happens in Julia, but on this course we avoid that possibility.

[3] And so do all other programming languagges.

[4] Julia recognises many more.

Logical operators

In week 1, we used truth tables to discuss the operators NOT, AND, OR. Let's now see how they work in Julia:

[DEMO: illustrate NOT, it is `!` via `!true` and `!false`]

[illustrate AND `&&` via `true && true`, `false && true`, `true && false` and `false && false`]

[illustrate OR `||` via `true ||true`, `false || true`, `true || false` and `false || false`]

Let's also illustrate the use of `( )` to make logical expressions unambiguous.

DEMO: `!true && false`
`!(true && false)`

In logical expressions, even more so than in arithmetical expressions, use brackets to make sure the expression is not ambigous.

Most often, we use the logical and comparison operators together, to make *logical tests*. These are expressions like $a > b$ OR $a < -b$, $x < 4$ AND $x > 0$, and NOT $x = 4$ (note the ambiguity with the equals sign here). Let's illustrate them all and more with a [DEMO:]

The truth values of these logical tests depend, of course, on the numerical values of the variables in them. When reading code, it is often necessary to answer questions like: for which values of `a` is the expression `a > -2 || a < 2` equal to `true` ?

Review and summary of Lesson

* The comparison operators on this course are `==`, `>`, `<`, `>=` and `<=`
* Using a comparison operator results in one of two values: `true` and `false`
* One can compare values of like type: numbers with numbers[5], characters with characters, and strings with strings
* The logical operators on this course are `!` for NOT, `&&` for AND and `||` for OR
* Logical tests often consist of names, comparison operators, logical operators and delimiters, and can become long[6]

[5] Note that this allows comparison across number types.
[6] Even on this course—see Week 4.