Lesson 1, Week 2: Arithmetic

Although, as we have said, this course is about text, not about numbers, some competence with arithmetic is essential in almost all programming projects.

AIM

— To describe and illustrate Julia's rules for arithmetic

After this lesson, you will be able to

* name and type the five arithmetic operators
* use parentheses to ensure that an arithmetic expression is not ambiguous
* combine values, names, operators and delimiters to make valid arithmetical expressions

REPL examples

In the REPL, one can just enter a valid arithmetic expression and its value is returned, e.g. `1+1`, `1 / 2 + 1`, `(5 - (25 - 9)^(1/2) )/2`.
(The latter can even be typed using the $\sqrt{}$ character)

[DEMO these expressions in the REPL, and many more.]

1 The arithmetic operators in Julia

Basic arithmetic in Julia uses the operators `+ - * /` (addition, subtraction, multiplication, division) and `^` (raising to a power).

Only left-right pairs of parentheses (round brackets) may be used as delimiters. In written and typeset arithmetic, other kinds of brackets often appear, but in Julia they throw errors inside arithmetic expressions.

Note that in Julia, the expression `-4` is meaningful[1].

[1] And so is `+4`, but of course we are not that interested in it.

2 How to ensure your arithmetic expression does what you want it to do

There is full set of rules in Julia, called "Operator precedence and associativity" that give a unique interpretation to the arithmetic of a given sequence of numbers and arithmetic operators. I recommend that you DON'T try to learn them at this stage. Instead, use brackets (round ones) to ensure that your arithmetic expression means what you want it to mean.

When an arithmetic expression is very long, making sure it has unique meaning requires a lot of brackets. For this reason, most programmers make frequent use of the rules of operator precedence to avoid using parentheses. However, on this course we don't use much arithmetic and when we do, the expressions are short. For that reason, we will insert many pairs of round brackets instead. In any case the rules are quite hard to learn and even experts make mistakes with long expressions. As a beginner, keep your arithmetic expressions short and use lots of brackets to make sure the rules are not needed.

The game: how many values can a sequence of numbers have?

Given a sequence of numbers like `1 4`, how many different arithmetic values can one create by inserting operators and delimiters?

For these two numbers, and no sign changes, the operators `+ - * /` create the values `5 -3 4` and `0.25`. Judicious use of minus symbols make it possible to create values with the opposite sign. That's eight, and `1^4` gives `1`, while `-(1^4)` gives -1. Ten values in all.

Does the sequence `4 1` give the same set of values? How does `2 3` compare?

Finally, try how many values you can get from the sequence `2 3 5` — note that every expression will have to use at least one pair of parentheses to ensure unique interpretation.

Using variable names to supply the values in an arithmetical expression

You do this just as you do in mathematics. For example, the expression `(2-3)*4` is equivalent to the lines

```
a, b, c = 2, 3, 4
(a-b)*c
```

There is more: the expression `2*a` can be replaced by `2a`. Similarly, `4*(2-3)` is equivalent to `4(2-3)`. The rule is: a number followed without whitespace by a name or by the start of parenthesis is interpreted as a multiplier—that is, a the multiply operator is inserted between the number and what follows it. Here's an example: the famous quadratic formula for the solution to $ax^2 + bx + c = 0$ gives two formulae for the roots, namely $x = (-b + (b^2 - 4ac)^{1/2})/2a$ and $x = (-b - (b^2 - 4ac)^{1/2})/2a$.

In Julia we can write them as
`root1 = (-b + (b^2 - 4a*c)^(1/2) )/2a` and
`root2 = (-b - (b^2 - 4a*c)^(1/2) )/2a` ;
note that the multiply operator is still necessary between `a` and `c`.

Review and summary

* Arithmetic expressions combine numbers and operators

* The arithmetic operators are `+` for adding, `-` for subtrating, `*` for multiplying, `/` for dividing and `^` for raising to a power.

* Arithmetic expressions are formed like in standard maths: start with a number, end with a number, with exactly one operator between adjacent numbers[2]

* Expressions with many numbers and operators can be ambiguous. Err on the side of caution: add enough `( )` pairs to ensure that only one interpretation of the computation is possible.

[2]For completeness: forms like `-7` use only one number, on the right of the operator.