Lesson 4, Week 3: Structures III
(`for ... end`)

AIM

— To learn to describe and use `for` loops

After this lesson, you will be able to

* Use a `for` code block to iterate over an iterable container
* Discuss the the local scope of a `for` with particular reference to the loop variable

`for` blocks have local scope

A `for` block introduces local scope: global variables can be read inside the block, but if their values are to change, they must be qualified with the keyword `global`. However, remember that you can replace a value in an array with another value of the same type[1].

Syntax of a `for` block by example

Our example is

```
for loopvar in 1:4
    println(('α' + loopvar)^loopvar)
end
```

The `println` function is called once for each value in the range.
The iteration requires the reserved keyword `for`, the operator `in` and an iterable value[a]

[a] In this case, the range `1:4`

You will notice that this is very similar to the comprehension `[('α'+ loc1)^loc1 for loc1 in 1:4]`, which creates exactly the same string values. This is because comprehension can be understood as a specialised kind of `for` loop: it uses the same keywords (except that a comprehension doesn't use `end`), it always produces an array, and it is one line of code only.

[1] For completeness: actually, any value that can be converted to the correct type will do, but we do not pursue that on this course.

Of course, the iterable supplying the values does not need to be a range. On this course, the iterable containers are strings, arrays and ranges, but Julia has many more. [DEMO]

You can change the value of the loop variable inside the loop, but this does not affect what value it takes on the next pass through the loop body. After all, it is local to the loop. But note that if you try to declare it global, Julia throws an error. [DEMO]

Pros and cons of `for` blocks vs `while` blocks vs comprehensions

Anything one does with a comprehension can be done with a `for` loop, anything that is done with a `for` loop can also be done with a `while` loop.

Question: if everything can be done with `while` code blocks, why the other two structures?

Answer: human convenience. High-level languages like Julia have one purpose only: to make things easier for those humans (like us!) who want to tell computers what we want them to do. You've seen that `while` loops can be infinite; they also take more lines to code than a `for` loop. For those two reasons, a `for` loop is better[2] if it can be used. Comprehension on the other hand is not necessarily better: it can for instance be harder to read when coming upon a piece of Julia code for the first time. The main reason for using a comprehension is that it takes up only one line, which can make code more compact. Overall, code is easier to read when it takes up less screen space.

Contrasting the code structures on this course

The structures we have seen are comprehensions, functions (two ways: with the keyword `function` and inline), logical tests using `if`, and the loops introduced by `while` and `for`.

- `if` blocks have global scope, the others have local scope.
- Except for comprehensions and inline functions, the structures start and end with a reserved keyword.
- Structures can be nested, but this possibility should not be over-used

Review and summary of Lesson

* `for` blocks have local scope
* The syntax is `for <loop variable name> in <iterable value> ...end`, where the `...` stands for the body of the loop
* The the loop variable takes the values in the iterable one by one and for each such value the body of the loop is executed
* The loop variable cannot be made visible globally

[2] From the human point of view, not the computer's, of course.