# Lesson 9, Week 1: Functions I

---

Functions in Julia is a very big subject, and in this first lecture we'll cover only a very little of it. In fact, we'll discuss only two built-in functions, but in the process we will consider multiple inputs, keywords and defaults. We will also start on the topic of inputs that differ by type[1], but only to a limited extent, reserving a more complete discussion for later in the course.

We will also introduce you to the the `?` way of using the REPL to query Julia's help system. You should plan to use this system early and often[2]!

## AIM

— To learn to use the functions `string` and `join`

— To understand the difference multiple inputs make to `join`

— To understand the difference that keywords make to `string` and the role of defaults

After this lesson, you will be able to

* Use `string` to combine several values into one string

* Use `string` to convert an integer into a string (optionally specifying a the base to use for the conversion)

* Use `join` to make a single string from a 1-dimensional array of strings (optionally specifying separators)

## `string` as method for combining values into a single string

We haven't seen multiple inputs to a function yet, here is the first example. Note that the comma is used as the delimiter—that is, the character that separates values from each other.

DEMO cases involving strings, characters and numbers, including variables with those as values, escape sequences, and `print`ing them.

---

[1]The word "type" has a Julian meaning that in this lecture we indicate very roughly. In a later lecture we consider Julia's type system in more detail.

[2]Until you don't need it, of course

## Calling `string` with an integer value allows some optional formatting

Let's turn to Julia's help system: simply enter `?` at the `julia>` prompt in the REPL. Notice that now the you get the `help?>` prompt. To get information, type a topic and hit Enter. [DEMO: help on `string`]

Interesting. It turns out that `string` does different things! Question: how does Julia decide? Anser: the pattern of inputs. In particular, whether or not you specify one or both of `base` and `pad`.

[DEMO: various possibilities with `base`, making sure to show some error messages]

We show this here to emphasise the use of the keyword `base` here. A keyword argument to a function is optional: if you say nothing, then Julia will use a default value, that is, Julia will act as if you gave the input `base=10`.

Now for the keyword `pad`. [DEMO]

## `join` as a method for combining strings, with optional formatting

In the preceding sections we saw a function that takes multiple separate inputs. But you could put values into a container and just pass that container to a function as a single value. Let's use `?` again to get help.

The following should work:

```
strarray = ["a", "bb", "ccc", "dddd"]
join(strarray, "; ", " but not ")
```

DEMO cases where not all the values are srings, some are characters and/or numbers

But there's more: the help tex on `join` has square brackets arount the last argument in the function call. This is means that it is optional—it can be left out. [DEMO]

## Keywords defaults vs optionally including some variables

Notice how `join` just concatenates an array of string values into a single string with no extra characters *unless* you pass it two or three arguments. That is, `join` behaves differently depending on the pattern of inputs. On the other hand, `string` changes behaviour when we include at least one of the keywords `base` or `pad`. In both cases, though, the behaviour you get depends on the pattern of inputs you supply. We will see this kind of thing again and again: most functions in Julia are generic, in the sense that they actually consist of many methods[3]. Which method you actually get when you call the function depends on the pattern of inputs you give.

DEMO: `methods(join)` reveals six methods, with six different patterns of input.

We will discuss generic functions and their patterns of input in more detail later in the course.

But why the difference? Couldn't `string` do things in the same way as `join`? Perhaps, but there is one obvious advantage that keywords have: if you omit some of them, the others have default values that are used. But `join`, because it doesn't use keywords with default values, doesn't allow us to omit the first delimiter and use only the second one.

For example, `join(strarray, , " and at last, ")` throws an error. We have to insert something as the first delimiter, even if it is just the empty string, in order to be able to set the second delimiter.

When you write your own functions, you may have to decide whether or not to use keywords. How to make the best decision is a relatively advanced topic we do not address in this course.

## Review and summary

* The function `string` when acting on many inputs of all kinds (separated by commas) concatenates them into a single string
* The function `string` can operate on just a single integer, where the keywords `base` and `pad` allow formatting to other than decimal numbers.
* The function `join` concatenates an array of strings into a single string
* The function `join` has alternate methods which allow delimiters to be inserted between the strings it concatenates, depending on how many values you put into its input list.

[3]From the point of view of computer science, this is one of the most interesting and unusual aspects of the Julia language, and many would say also one of its greatest strengths.