Juyst

A tutorial

---

1. *Getting started*

Since Juyst bridges between Julia and Typst, we also have to get two things running. First, install the Julia package `Juyst` from the general registry by executing

```
julia> ]

(@v1.10) pkg> add Juyst
```

You only have to do this once. (It is like installing and using the Pluto notebook system, if you are familiar with that.)

When you want to use Juyst in a Typst document (say, `your-document.typ`), add the following line at the top:

```
#import "@preview/juyst:0.1.0": *
```

Then, open a Julia REPL and run

```
julia> import Juyst

julia> Juyst.run("your-document.typ")
```

Juyst facilitates the communication between Julia and Typst via a CBOR file. This is like JSON or TOML but consists of binary data rather than text so it allows to store, for example, images. By default, Juyst uses the name of your document and adds a `-juyst.cbor`, so `your-document.typ` would become `your-document-juyst.cbor`. This can be configured, of course.

To let Typst know of the computed data in the CBOR file, add the following line to your document:

```
#read-julia-output(cbor("your-document-juyst.cbor"))
```

By first running the Julia component of Juyst before compiling the Typst document, you ensure that the CBOR file exists and Typst doesn't immediately throw an error.

You are now ready to go! The running Julia function watches your file and performs the necessary computations whenever you save it (very similar to `typst watch`).

2. *The `jl` function*

This is the most important function when using Juyst in your document. `jl` takes a piece of Julia code and the result of that code is inserted in the document.

Let's start with a very simple example:

```
#jl(```julia
  greeting = rand(["Hello", "Hi", "Good morning"])
  "$greeting, this is Julia in Typst via Juyst!"
```)
```

This produces:

"Hi, this is Julia in Typst via Juyst!"

Try adding some more content to your document without changing the Julia code and save the document. You will notice that Juyst recognises the Julia code has not changed and thus does not rerun it! This is of course very important when you are writing a long document.

## 3. *Package management*

Most non-trivial Julia code will use external packages. To specify what Julia packages you want to import, you can use the `jl-pkg` function in Typst. It accepts an arbitrary amount of string arguments where each of them specifies a Julia package in [the same way as you would in the Julia package REPL](https://pkgdocs.julialang.org/v1/repl/#repl-add).

Let's try it out!

```
#jl-pkg("Example@0.4", "Plots")

What is $3 + 5$?

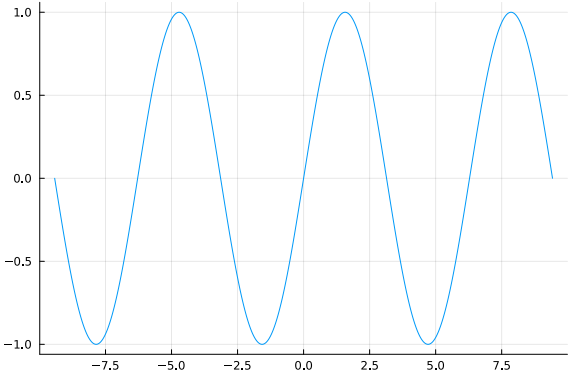
#jl(```julia
  import Example
```)
```

```
  Example.domath(3)
```')

Let's plot something!

```
#set image(width: 20em)
#jl(```julia
  using Plots
  plot(pi .* (-3:.01:3), sin, legend = nothing)
```')
```

What is $3 + 5$?

8

Let's plot something!




log Precompiling Plots [91a5bcdd-55d7-5caf-9e0b-520d859cae80]
```