Julia, not Juliet

an introduction to the programming language, not Shakespeare

Kamillo Ferry

Technische Universität Berlin

July 3rd, 2024

OSCAR Boot Camp

What is Julia?

- Work on Julia began in 2009.
- The syntax is being considered stable since 2018.
- 'Julia was designed for high performance [...] and automatically compiles to native code [...]',
- meaning, while being compiled, we also get some advantages of a dynamic language.

How to get Julia onto your device?

Most Unix-like OSs:
`curl -fsSL https://install.julialang.org | sh` as per the instructions on the official Julia page (`julialang.org`).

Windows:
Possible. But you will want Julia on Linux here, so rather take the Windows Subsystem for Linux route. (If you need help, ask me afterwards.)

How to get Julia actually working?

In general: You want to have a working and recent compiler.

Debian-based:

```
sudo apt-get install build-essential
```

Fedora:

```
sudo dnf install gcc-c++ make
```

macOS:

```
xcode-select --install
```

Time to see Julia in action!

What is so nice about Julia (besides OSCAR)?

- It's interactive!
- Broadcasting is very nice in Julia
- Multiple dispatch

A first function in Julia

Writing a function that operates on vectors:

```
function add(x::Vector, y::Vector)
    n = length(x)
    z = Vector()
    for i in 1:n
        push!(z, x[i]+y[i])
    end

    return z
end
```

A first function, second take

Taking advantage of broadcasting:

```
add(x, y) = x+y

add(x::Vector, y::Vector) = add.(x,y)
```

A first function, last take

```
x .+ y   // broadcasting

x + y   // Vector-defined operator
```

What is so nice about Julia (besides OSCAR)?

- It's interactive!
- Broadcasting is very nice in Julia
- Multiple dispatch

A first function, second take

Taking advantage of broadcasting:

```
add(x, y) = x+y

add(x::Vector, y::Vector) = add.(x,y)
```

Multiple dispatch, extended

Extending add and making it more polymorphic:

```
add(x,y) = x+y

add(x::Vector, y::Vector) = add.(x,y)
```

Now we extend add somewhere else:

```
add(P::Polyhedron, Q::Polyhedron) = minkowski_sum(P,Q)
```

There's also a parameter template system:

```
*(c::T, A::S) where {T <: RingElem,
                     S <: MatElem{T}}
```

How to navigate around all that?

- Using the help prompt mode
- `methodswith(::Type; supertypes=true)`
- `@less` and `@which`
- Tab completion with some arguments filled in already

Conditional branching

```
function do_branching(n::Integer)
  if n%2 == 0
    return n/2
  elseif n%2 == 1
    return 3*n + 1
  end
end
```

Conditional branching

```
function do_branching(n::Integer)
  if n%2 == 0
    return n/2
  else
    return 3*n + 1
  end
end
```

Conditional branching

```
function do_branching(n)
  if n%2 == 0
    return n/2
  else
    return 3*n + 1
  end
end
```

Unbounded loops

```
function euclid(x,y)
  while x!=y
    if x > y
      x -= y
    else
      y -= x
    end
  end
  return x
end
```

Working with more complex data structure

Suppose we want to implement a data structure in Julia that describes a permutation.

- The permutation should be specified by the images.
- A permutation σ should work with function-call syntax, i. e. we can write $\sigma(i)$ and it works.
- Composition of permutations should work.

A problem in elimination theory

Suppose we have the polynomial map given by

$$\begin{aligned}\varphi\colon k &\to k^3,\\ x &\mapsto ((1-x)^2, 2x(1-x), x^2).\end{aligned}$$

How do we calculate the relations of $\overline{\operatorname{im}\varphi}$ in `OSCAR`?

Complete independence models

- Suppose we have two finite random variables X_1, X_2 with values in $\{1, \ldots, n_1\}$ and $\{1, \ldots, n_2\}$ each.
- Then, the random vector (X_1, X_2) can be characterized by a stochastic matrix $P \in \mathbb{R}^{n_1 \times n_2}$.
- We know that $X_1 \perp\!\!\!\perp X_2$ if all 2×2-minors of P vanish.
- *How to check this in* `OSCAR`*?*

Beyond base Julia

- This means we want to install packages.
- There's two ways to do this, e. g. for `OSCAR`
  1. Enter `using Pkg; Pkg.add("OSCAR")` into the Julia REPL.
  2. Hit `]` on an empty line and enter `add OSCAR`.

Editors

There are a plenty of. Emacs, Vim and VS Code are officially supported.

- **Emacs**: `julia-emacs` together with `julia-repl` and `eshell`
- **Vim**: `julia-vim` together with `vim-slime` and `tmux`
- **VS Code**: official Julia extension for VS Code

  With the above, you get syntax highlighting and can send code from the editor directly to a REPL

Running code from disk

- Maybe typing code directly into the terminal is not always comfortable.
- Run a local file like a script: `julia file.jl`
- Run a file inside a Julia session: `include("file.jl")`

Notebooks

Beyond plain text editors, there are even two options for interactive notebooks, Pluto and Jupyter.

- Getting either is as easy as running `]add Pluto` or `]add IJulia`
- Running Pluto: `using Pluto; Pluto.run()`
- Running Jupyter: `using IJulia; notebook()`

Now go explore the world with Julia and `OSCAR`!