# Extensible Software for Research

## Principles and an Example in julia

Center for Lifespan Psychology, Max Planck Institute for Human Development

Maximilian Ernst

# Contents

- Why should you care?
- How do you get there?

# A day in the life of …

a research software developer

# A day in the life of …

a research software developer

research software engineers - research software developers - applied users

# A day in the life of …

a research software developer

research software engineers - research software developers - applied users

- work with a specific type of model
  - linear regression, deep learning, …

# A day in the life of …

a research software developer

research software engineers - research software developers - applied users

- work with a specific type of model
  - linear regression, deep learning, …
- have an idea

# A day in the life of …

a research software developer

research software engineers - research software developers - applied users

- work with a specific type of model
  - linear regression, deep learning, …
- have an idea
- test it

# A day in the life of …

a research software developer

research software engineers - research software developers - applied users

- work with a specific type of model
  - linear regression, deep learning, …
- have an idea
- test it
- make it available to applied researchers

# Now we need software

- to test $\rightarrow$ prototype
- to make it available $\rightarrow$ deploy

# Now we need software

- to test $\rightarrow$ prototype
- to make it available $\rightarrow$ deploy

What's the fastest way to get there?

# Now we need software

- to test $\rightarrow$ prototype
- to make it available $\rightarrow$ deploy

What's the fastest way to get there?

They are already using existing software.

# Now we need software

- to test $\rightarrow$ prototype
- to make it available $\rightarrow$ deploy

What's the fastest way to get there?

They are already using existing software.

**It would be nice if they could extend existing software!**

# But …

# But …

- **understand** 1000s of lines of code

# But …

- **understand** 1000s of lines of code
- make **changes**, possibly breaking stuff

# But …

- **understand** 1000s of lines of code
- make **changes**, possibly breaking stuff
- get maintainers to **adopt** their changes

# But …

- **understand** 1000s of lines of code
- make **changes**, possibly breaking stuff
- get maintainers to **adopt** their changes

**These hurdles are often too high!**

# A day in the life of …

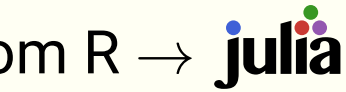
# A year in the life of …

- to test: minimal reimplementation
  - waste of time
  - not well tested
  - harder to reproduce
  - slow

# A year in the life of …

- to test: minimal reimplementation
  - waste of time
  - not well tested
  - harder to reproduce
  - slow
- to deploy: put code on github
  - bad user interface, no documentation
  - missing features
  - incompatible to existing software

# My Experience

- from R → 

# Culture

- care about extensibility
- developer documentation
- assume that code is read

# Software Design

You need to be able to add new features...

- without **understanding** existing code
- without **changing** existing code
- **syntactical requirements** need to be clear

# An example: time on a clock

- ranges from $0 - 11:59$
- $11 + 6 = 5$
- $3 \cdot 6 = 6$
- for simplicity: $5.5 = 5:30$

# An example: time on a clock

- ranges from $0 - 11:59$
- $11 + 6 = 5$
- $3 \cdot 6 = 6$
- for simplicity: $5.5 = 5:30$

disclaimer: I won't show the best way to implement this in julia, but the most instructive way that we can do in a few minutes!

# Everything has a type

```
a = 1.0
typeof(a) # Float64

b = ”hello”
typeof(b) # String
```

# Multiple dispatch

```
6.6*7.9

methods(*)
# 364 methods for generic function "*":
# ...
# [56] *(x::Number, A::LinearAlgebra.UpperTriangular)
# in LinearAlgebra at /usr/share/julia/stdlib/v1.7/
# LinearAlgebra/src/triangular.jl:859
# ...
# [350] *(z::Complex, x::Real) in Base at complex.jl:334

@which 6.6*7.9
# *(x::Float64, y::Float64) in Base at float.jl:405
@which 6*7
# *(x::T, y::T) where T<:Union{..., Int64, ...}
# in Base at int.jl:88
```

# You can define your own types…

```
struct ClockTime
    time
end

my_time = ClockTime(5.0)
my_time.time # 5.0
```

# …and methods:

```
import Base: +, *

function +(x::ClockTime, y::ClockTime)
    return ClockTime((x.time + y.time) % 12)
end

function *(x::Real, y::ClockTime)
    return ClockTime((x * y.time) % 12)
end

my_time = ClockTime(11.2)
your_time = ClockTime(5.3)

our_time = my_time + your_time # ClockTime(4.5)
7*my_time # ClockTime(6.39...)
```

# Sparse matrices of clocktimes

```
a = zeros(20, 20)
a[3,9] = 1
a[6,9] = sqrt(2)
a[19,1] = π
a[4,5] = ℯ

b = reshape(fill(ClockTime(0.0), 400), 20, 20)
b[1,1] = ClockTime(5.0)
b[1,2] = ClockTime(11.75)
b[6,9] = ClockTime(1.4)
b[16,4] = ClockTime(7.8)
```

- $(ab)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

# Sparse matrices of clocktimes

```
using SparseArrays

a_sparse = sparse(a)

b_sparse = sparse(b)
# ERROR: MethodError:
# no method matching zero(::ClockTime)

import Base: zero

zero(x::ClockTime) = ClockTime(zero(x.time))

b_sparse = sparse(b)
```

# Sparse matrices of clocktimes

```
using BenchmarkTools

@benchmark a_sparse*b_sparse

BenchmarkTools.Trial: 10000 samples with 183 evaluations.
 Range (min … max):  571.831 ns …  21.539 μs
 Time  (median):     635.415 ns
 Time  (mean ± σ):   821.828 ns ± 932.458 ns
Memory estimate: 2.31 KiB

@benchmark a*b

BenchmarkTools.Trial: 6840 samples with 1 evaluation.
 Range (min … max):  611.887 μs …   3.603 ms
 Time  (median):     686.239 μs
 Time  (mean ± σ):   727.364 μs ± 221.404 μs
Memory estimate: 678.25 KiB
```

# Recap

- functions (in the SparseArrays package) are composed of other functions
- if I can provide the appropriate methods for my type, their code also works for me
- they can give me a **list of methods**, and I do not need to **understand** or **modify** their code

# Why julia? - the bottom line

Because functions can implement very abstract behaviour, this mode of extensibility is widely applicable:

- there can be packages that work with flowers, spaceships or loss functions
- that depends on the scent, speed or gradient
- If I can add a new type and write the appropriate methods, it will work for me!

Thanks!