“  ”

A Fast Dynamic Language for Technical Computing

Created by: Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman et. al.

A sane and friendly programming language which allows

- to write clean high level code
- and do fast low level number crunching

in one language within one framework.

Some stated design principles

- Open source with an MIT licensed core
- Dynamically typed with fast user-defined types
- Multiple dispatch combined with a parametric type system
- JIT compiler - fast vectorized and fast iterative code
- Metaprogramming
- Single environment to do technical computing and surrounding general programming tasks

Example: Running average

```
1  X = cumsum(randn(10^6)) #random walk
2
3  # running average of order three, interactive style
   runavg3(X) = [ (X[i-1] + X[i] + X[i+1])/3 for
4      i=2:length(X)-1 ]
5
6  function runavg(X, d) #first shot at a generalization
7      n = length(X)
8      Y = similar(X, n - d + 1)
9      Y[1] = mean(X[1:d])
10     for i in 2:(n-d+1)
11         Y[i] = Y[i-1] + 1/d * (X[i-1+d] - X[i-1])
12     end
13     Y
14 end
```

Julia unimposingly computes the result very fast:

```
julia> @elapsed runavg3(X)
0.003496773

julia> @elapsed runavg(X,3)
0.004001902

julia> @elapsed runavg(X,30)
0.004068611

julia> @elapsed cumsum(X)
0.004236988
```

(Most of this is allocating the new array.)

Type system

Two uses

- Dynamic: Using types to dispatch the right method at runtime

```
expm(A::HermOrSym) = (F = eigfact(A);
    F.vectors*Diagonal(exp(F.values))*F.vectors')
```

- (Quasi) static: Helping the just-in-time compiler

```
julia> xs = 1:5
julia>[i^3 for i in xs]
5-element Array{Any,1}
julia>[i::Int^3 for i in xs]
5-element Array{Int64,1}
```

Method dispatch

```
#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))
```

You remember?

Static languanges: Multiple dispatch, can follow a elaborated pattern, at compile time (function overloading.)

Dynamic languanges: Single dispatch or no dispatch, simple, at run time.

Julia: Elaborated, multiple dispatch with promotion rules, but does not slow down JIT'ed code.

Function overloading in C++

You would not want to do this on runtime...

```
Argument-matching conversions occur in the following
    order:
An exact match, in which the actual arguments exactly
    match the type and number of formal arguments of one
    declaration of the overloaded function. This includes
    a match with one or more trivial conversions.
A match with promotions in which a match is found when one
    or more of the actual arguments is promoted
A match with standard conversions in which a match is
    found when one or more of the actual arguments is
    converted by a standard conversion
A match with user-defined conversions in which a match is
    found when one or more of the actual arguments is
    converted by a user-defined conversion
A match with ellipses
```

(OS/390 V2R10 C/C++ Language Reference)

Multiple dispatch in Julia

```
julia> methods(max)
# 14 methods for generic function "max":
max(x::Float64,y::Float64) at math.jl:334
max(x::Float32,y::Float32) at math.jl:335
max(x::BigFloat,y::BigFloat) at mpfr.jl:509
max{T<:Real}(x::T<:Real,y::T<:Real) at promotion.jl:191
max(x::Real,y::Real) at promotion.jl:172
max{T1<:Real,T2<:Real}(x::T1<:Real,y::AbstractArray{T2<:Real,N})
    at operators.jl:247
max{T1<:Real,T2<:Real}(x::AbstractArray{T1<:Real,N},y::T2<:Real)
    at operators.jl:249
max{T1<:Real,T2<:Real}(x::AbstractArray{T1<:Real,N},y::AbstractA
    at operators.jl:253
max(x,y) at operators.jl:35
max(a,b,c) at operators.jl:67
max(a,b,c,xs...) at operators.jl:68
```

Metaprogramming

```
1  for (fname, felt) in ((:zeros,:zero),
2                        (:ones,:one),
3                        (:infs,:inf),
4                        (:nans,:nan))
5  @eval begin
   ($fname){T}(::Type{T}, dims...)   = fill!(Array(T,
6      dims...), ($felt)(T))
   ($fname)(dims...) = fill!(Array(Float64, dims...),
7      ($felt)(Float64))
   ($fname){T}(x::AbstractMatrix{T}) = ($fname)(T, size(x,
8      1), size(x, 2))
9  end
10 end
```

```
julia> ones(3)
5-element Array{Float64,1}:
 1.0
 1.0
 1.0
```

Word of caution

- Young language, initial commit 2009, open source 2012
- “Fast moving target“
- Memory hungry: Compiled and specialized methods
- Limited debugging support