The Julia Manifesto

Jonathon LeFaive
CSG Tech Talk
June 30, 2016
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
MIT Licensed

Free and Open Source
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
# LLVM-based JIT Compiler

<table>
<thead>
<tr>
<th></th>
<th>Fortran</th>
<th>Julia</th>
<th>Python</th>
<th>R</th>
<th>Matlab R2015b</th>
<th>Octave 4.0.0</th>
<th>Mathematica 10.2.0</th>
<th>JavaScript V8 3.28.71.19</th>
<th>Go g01.5</th>
<th>LuaJIT gsl-shell 2.3.1</th>
<th>Java 1.8.0_45</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib</td>
<td>0.70</td>
<td>2.11</td>
<td>77.76</td>
<td>533.52</td>
<td>26.89</td>
<td>9324.35</td>
<td>118.53</td>
<td>3.36</td>
<td>1.86</td>
<td>1.71</td>
<td>1.21</td>
</tr>
<tr>
<td>parse_int</td>
<td>5.05</td>
<td>1.45</td>
<td>17.02</td>
<td>45.73</td>
<td>802.52</td>
<td>9581.44</td>
<td>15.02</td>
<td>6.06</td>
<td>1.20</td>
<td>5.77</td>
<td>3.35</td>
</tr>
<tr>
<td>quicksort</td>
<td>1.31</td>
<td>1.15</td>
<td>32.89</td>
<td>264.54</td>
<td>4.92</td>
<td>1866.01</td>
<td>43.23</td>
<td>2.70</td>
<td>1.29</td>
<td>2.03</td>
<td>2.60</td>
</tr>
<tr>
<td>mandel</td>
<td>0.81</td>
<td>0.79</td>
<td>15.32</td>
<td>53.16</td>
<td>7.58</td>
<td>451.81</td>
<td>5.13</td>
<td>0.66</td>
<td>1.11</td>
<td>0.67</td>
<td>1.35</td>
</tr>
<tr>
<td>pi_sum</td>
<td>1.00</td>
<td>1.00</td>
<td>21.99</td>
<td>9.56</td>
<td>1.00</td>
<td>299.31</td>
<td>1.69</td>
<td>1.01</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>rand_mat_stat</td>
<td>1.45</td>
<td>1.66</td>
<td>17.93</td>
<td>14.56</td>
<td>14.52</td>
<td>30.93</td>
<td>5.95</td>
<td>2.30</td>
<td>2.96</td>
<td>3.27</td>
<td>3.92</td>
</tr>
<tr>
<td>rand_mat_mul</td>
<td>3.48</td>
<td>1.02</td>
<td>1.14</td>
<td>1.57</td>
<td>1.12</td>
<td>1.12</td>
<td>1.30</td>
<td>15.07</td>
<td>1.42</td>
<td>1.16</td>
<td>2.36</td>
</tr>
</tbody>
</table>

**Figure:** benchmark times relative to C (smaller is better, C performance = 1.0).
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
julia> e = parse("1 + 2 * 3")
:(1 + 2 * 3)
julia> dump(e)
Expr
  head: Symbol call
  args: Array(Any,(3,))
    1: Symbol +
    2: Int64 1
    3: Expr
      head: Symbol call
      args: Array(Any,(3,))
        1: Symbol *
        2: Int64 2
        3: Int64 3
      typ: Any
      typ: Any
julia> Meta.show_sexpr(e)
(:call, :+, 1, (:call, :*, 2, 3))
julia> e2 = Expr(:call, :+, 1, Expr(:call, :*, 2, 3))
:(1 + 2 * 3)
julia> e == e2
true
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Operators

\[ \begin{align*}
1 + 2 & \implies 3 \\
11 - 1 & \implies 10 \\
10 \times 3 & \implies 30 \\
35 / 5 & \implies 7.0 \\
5 \div 2 & \implies 2.5 \\
5 \div 2 & \implies 2.5 \\
2 \hat{} 2 & \implies 4 \quad \text{# power not xor} \\
13 \% 10 & \implies 3 \\
2 < 3 \land 2 & \implies \text{false} \\
2 \times \pi & \implies 6.283185307179586 \\
1 \in [1, 3, 4] & \implies \text{true} \\
1 \notin [1, 3, 4] & \implies \text{false} \\
[1, 2] \cup [3, 4, 5] & \implies [1, 2, 3, 4, 5]
\end{align*} \]

# Custom unicode function
\[ \sum(x, y) = x + y \implies \sum \quad \text{(generic function with 1 method)} \\
\sum(1, 2) & \implies 3 \]

# Unicode alias
\[ \sum = + \implies + \quad \text{(generic function with 171 methods)} \\
\sum(1, 2, 3, 4) & \implies 10 \]

# Fraction
\[ \text{typeof}(2 \div 5) & \implies \text{Rational\{Int64\}} \]
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Control Flow

# create a variable
foo = 2

# if statement
if foo < 5
    println("Is less than 5.")
elseif foo > 5
    println("Is greater than 5.")
else
    println("Is 5.")
end

# iterate over an array
for color in ["red", "green", "blue"]
    println(color)
end
Accessing Arrays

arr = [1, 2, 3, 4, 5]

arr[1] # => 1

arr[end] # => 5


arr[2:end] # => [2, 3, 4, 5]

for i = 1:length(arr)
    println(arr[i])
end
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Number of Packages by Julia Version
Working with Data

```julia
julia> using GLM, RDatasets

julia> form = dataset("datasets","Formaldehyde")
6x2 DataFrame
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Row #</td>
<td>Carb</td>
<td>OptDen</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.086</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
<td>0.269</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.446</td>
</tr>
<tr>
<td>4</td>
<td>0.6</td>
<td>0.538</td>
</tr>
<tr>
<td>5</td>
<td>0.7</td>
<td>0.626</td>
</tr>
<tr>
<td>6</td>
<td>0.9</td>
<td>0.782</td>
</tr>
</tbody>
</table>

julia> lm1 = fit(LinearModel, OptDen ~ Carb, form)
Formula: OptDen ~ Carb

Coefficients:

| Estimate | Std. Error | t value | Pr(>|t|) |
|----------|------------|---------|----------|
| Intercep | 0.00508571 | 0.00783368 | 0.649211 | 0.5516 |
| Carb     | 0.876286   | 0.0135345  | 64.7444  | 3.4e-7 |

julia> confint(lm1)
2x2 Array{Float64,2}:
-0.0166641  0.0268355
0.838708   0.913864
```
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Strings

c = \u2200
typeof(c) #=> Char
Int(c) #=> 8704

str = "foobar"
typeof(str) #=> ASCIIString
ustr = UTF8String("foobar")
typeof(ustr) #=> UTF8String
ustr2 = "foobar \u2200"
typeof(ustr2) #=> UTF8String

s = "1 + 2 = $(1 + 2)" #=> "1 + 2 = 3"
world = "Earth"
msg = "Hello $(world)!" #=> "Hello Earth!"

m = match(r"(a|b)(c)?(d)", "acd") #=> RegexMatch("acd", 1="a", 2="c", 3="d")
m[1] #=> "a"
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Built-in Linear Algebra

<table>
<thead>
<tr>
<th>Matrix type</th>
<th>+</th>
<th>-</th>
<th>*</th>
<th>\</th>
<th>Other functions with optimized methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermitian</td>
<td>MV</td>
<td></td>
<td></td>
<td></td>
<td>(\text{inv}(), \text{sqrtm}(), \text{expm}())</td>
</tr>
<tr>
<td>UpperTriangular</td>
<td>MV</td>
<td>MV</td>
<td></td>
<td></td>
<td>(\text{inv}(), \text{det}())</td>
</tr>
<tr>
<td>LowerTriangular</td>
<td>MV</td>
<td></td>
<td>MV</td>
<td></td>
<td>(\text{inv}(), \text{det}())</td>
</tr>
<tr>
<td>SymTridiagonal</td>
<td>M</td>
<td>M</td>
<td>MS</td>
<td>MV</td>
<td>(\text{eigmax}(), \text{eigmin}())</td>
</tr>
<tr>
<td>Tridiagonal</td>
<td>M</td>
<td>M</td>
<td>MS</td>
<td>MV</td>
<td></td>
</tr>
<tr>
<td>Bidiagonal</td>
<td>M</td>
<td>M</td>
<td>MS</td>
<td>MV</td>
<td></td>
</tr>
<tr>
<td>Diagonal</td>
<td>M</td>
<td>M</td>
<td>MV</td>
<td>MV</td>
<td>(\text{inv}(), \text{det}(), \text{logdet}(), /())</td>
</tr>
<tr>
<td>UniformScaling</td>
<td>M</td>
<td>M</td>
<td>MVS</td>
<td>MVS</td>
<td>/()</td>
</tr>
</tbody>
</table>

**M (matrix)**: An optimized method for matrix-matrix operations is available

**V (vector)**: An optimized method for matrix-vector operations is available

**S (scalar)**: An optimized method for matrix-scalar operations is available

- **Cholesky**: Cholesky factorization
- **CholeskyPivoted**: Pivoted Cholesky factorization
- **LU**: LU factorization
- **LUTridiagonal**: LU factorization for Tridiagonal matrices
- **UmfpackLU**: LU factorization for sparse matrices (computed by UMFPack)
- **QR**: QR factorization
- **QRCOMPACTWY**: Compact WY form of the QR factorization
- **QRpivot**: Pivoted QR factorization
- **Hessenberg**: Hessenberg decomposition
- **Eigen**: Spectral decomposition
- **SVD**: Singular value decomposition
- **GeneralizedSVD**: Generalized SVD
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Running External Programs

julia> run(pipeline(`cut -d: -f3 /etc/passwd`, `sort -n`, `tail -n5`))

julia> names = ["foo", "bar", "baz"]
3-element Array{ASCIIString,1}:
    "foo"
    "bar"
    "baz"

julia> exts = ["aux", "log"]
2-element Array{ASCIIString,1}:
    "aux"
    "log"

julia> `rm -f $names.$exts`
`rm -f foo.aux foo.log bar.aux bar.log baz.aux baz.log`
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
Multiple Dispatch and Parametric Types

my\_func(x, y) = 2x + y
my\_func(1, 2) # => 4
my\_func('a', 'b') # => ERROR

function my\_func(x::Char, y::Char)
    return 2 * Int64(x) + Int64(y)
end
my\_func('a', 'b') # => 292

type Point
    x
    y
    z
end
Point(1, 3.0, "foobar")

type Point\{T\}
    x::T
    y::T
    z::T
end
Point(1, 2, 3)

type Point\{T, T2\}
    x::T
    y::T
    z::T2
end
Point(1, 2, 3.0)

type Point
    x::Float64
    y::Float64
    z::Float64
end
Point(1.0, 2.0, 3.0)
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. **We want it interactive and we want it compiled.**

(Did we mention it should be as fast as C?) ...
Why They Created Julia

... We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?) ...
```julia
function loop_sum(arr)
    ret = 0.0
    for val in arr
        ret += val
    end
    return ret
end
```
loop_sum <- function(arr) {
  ret <- 0.0
  for (val in arr) {
    ret <- ret + val
  }
  return(ret)
}
double loop_sum(const std::vector<double>& arr) {
    double ret = 0.0;
    const std::size_t arr_size = arr.size();
    for (std::size_t i = 0; i < arr_size; ++i) {
        ret += arr[i];
    }
    return ret;
}
function loop_sum(arr::Array{Float64,1})
    ret::Float64 = 0.0
    for i = 1:length(arr)
        ret += arr[i]
    end
    return ret
end
Further Reading

https://learnxinyminutes.com/docs/julia/

http://docs.julialang.org/en/release-0.4/stdlib/c/

http://docs.julialang.org/en/release-0.4/stdlib/parallel/

http://docs.julialang.org/en/release-0.4/manual/modules/