What’s up with Julia?

Galen O’Neil
Physicist, NIST, Boulder, CO
My background

Pulse height $\sim =$ energy/heat capacity
Oversimplification

- C and FORTRAN are static and fast.
- Python, R, and MATLAB are dynamic and slow.
  - But it’s ok because the functions are written in C or FORTRAN so they’re fast. AKA vectorize everything AKA the two language problem.
- Julia is trying to be dynamic and fast. One language.
How fast?

Rough goal: should be able to get within \(\sim 2\) of \(c\).
How fast?

Rough goal: should be able to get within ~2 of c.
Randmatstat code
Python, Julia, FORTRAN

### Python

```python
def mandel(z):
    maxiter = 80
    c = z
    for n in range(maxiter):
        if abs(z) > 2:
            return n
        z = z*z + c
    return maxiter
```

### FORTRAN

```fortran
integer function mandel(z0) result(r)
complex(dp), intent(in) :: z0
complex(dp) :: c, z
integer :: n, maxiter
maxiter = 80
z = z0
Do n = 1, maxiter
   if (abs(z) > 2) then
      r = n-1
      return
   end if
   z = z**2 + c
end do
r = maxiter
end function
```

### Julia

```julia
function mandel(z)
    c = z
    maxiter = 80
    for n = 1:maxiter
        if abs(z) > 2
            return n-1
        end
        z = z^2 + c
    end
    return maxiter
end
```
How is Julia so fast?
How is can Julia be so fast?

Types

- Type inference: If the compiler knows the types of the variables, it can efficiently use the hardware.

- Type Stability: A function’s return type should depend only on the input types (not values!)

```julia
julia> sqrt(1)
1.0

julia> sqrt(-1) # MATLAB would return 0+1i here
ERROR: DomainError
sqrt will only return a complex result if called with a complex argument.
try sqrt(complex(x))
in sqrt at math.jl:131
```
Anything else?

- MIT licensed, free and open source!
- Built in package manager.
- ipython like REPL out of the box.
- Easy to call python.

```julia
julia> Pkg.add("PyCall")
INFO: Cloning cache of PyCall from git://github.com/stevengj/PyCall.jl.git
INFO: Installing PyCall v0.4.10
INFO: Package database updated
julia> using PyCall
julia> @pyimport scipy.signal as ss
julia> ss.gaussian(10,1)'
```

```
1x10 Array{Float64,2}:
 4.00653e-5  0.00218749  0.0439369  0.324652 ...  0.0439369
 0.00218749  4.00653e-5
```
Anything else?

- Parallelism
- Profiler
- Mostly (70%+) written in Julia.
- Unicode.

```julia
julia> a = rand(100);
julia> y = map(sin,a);

julia> y_p = pmap(sin,a); # this way uses multiple processes if you started with -p flag

julia> y==y_p
true

julia> function α(x)
    s=0.0
    for j=1:1000
        s+=sin(x*j)
    end
    s
end

α (generic function with 1 method)

julia> @profile for j=1:100000
    α(j) end

julia> Profile.print()
```

```
2     client.jl; _start; line: 399
2 REPL.jl; run_repl; line: 170
2 REPL.jl; run_frontend; line: 818
2 LineEdit.jl; run_interface; line: 1379
1 REPL.jl; anonymous; line: 585
1 REPL.jl; send_to_backend; line: 571
1 REPL.jl; send_to_backend; line: 574
1 REPL.jl; anonymous; line: 586
56526 task.jl; anonymous; line: 96
56526 REPL.jl; eval_user_input; line: 54
56524 profile.jl; anonymous; line: 1
30494 none; f; line: 4
26028 none; α; line: 4
1    none; α; line: 6
1    profile.jl; anonymous; line: 12
1    profile.jl; anonymous; line: 14
```
Anything else?

- Your types are equal to built in types.
- Multiple dispatch.
- Easy to call C.
- Just in time compiling.
Python to Julia Example

### Python

```python
code:import numpy, time
code:r = numpy.random.rand(10000000)
code:
def summarize(r):
    code:return r.mean(), numpy.amax(r), numpy.amin(r)
code:
def summarize_loop(r):
    code:maxval = -numpy.inf
    code:minval = numpy.inf
    code:s = 0.0
    code:for i in range(len(r)):
        code:d = r[i]
        code:s+=d
        code:if d > maxval:
            code:maxval = d
        code:elif d < minval:
            code:minval = d
    code:return s/len(r), maxval, minval

code:tstart = time.time()
code:meanval, maxval, minval = summarize_loop(r)
code:tend=time.time()
code:print(10*(tend-tstart))

code:tstart = time.time()
code:for i in range(10):
    code:meanval, maxval, minval = summarize(r)
code:tend=time.time()
code:print(tend-tstart)
```

### Julia

```julia
code:r = rand(10000000)
code:
s = summarize(r) = mean(r), maximum(r), minimum(r)
code:
function summarize_loop(r)
    code:maxval = realmin(eltype(r))
    code:minval = realmax(eltype(r))
    code:s = zero(eltype(r))
    code:for i = 1:length(r)
        code:d = r[i]
        code:s+=d
        code:if d > maxval
            code:maxval = d
        code:elseif d < minval
            code:minval = d
        code:end
    code:end
    code:return s/length(r), maxval, minval

code:@time for i=1:10 summarize(r) end
@time for i=1:10 summarize_loop(r) end
```

### Comparison Table

<table>
<thead>
<tr>
<th>Method</th>
<th>Python Time</th>
<th>Julia Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorized</td>
<td>54 ms</td>
<td>54 ms</td>
</tr>
<tr>
<td>Loop</td>
<td>7400 ms</td>
<td>31 ms</td>
</tr>
</tbody>
</table>
What’s the catch?

• Julia just released version 0.3. It is a new and young (v0.1 released Feb 2013) language under rapid development. Breaking changes will happen.

• Package ecosystem isn’t at the python or R level, but it’s pretty good and growing fast.

• IDE support is in the early stages.

• Upcoming in version 0.4 (aka currently missing)

  • Debugger

  • Array views by default (aka a=b[5:500] will be fast because it doesn’t make a copy)

  • Static compilation

  • Faster strings
What’s the catch?

```plaintext
function summarize_loop(r)
    maxval = realmin(eltype(r))
    minval = realmax(eltype(r))
    s = zero(eltype(r))
    for i = 1:length(r)
        d = r[i]
        s+=d
        if d > maxval
            maxval = d
        elseif d < minval
            minval = d
        end
    end
    s/length(r), maxval, minval
end

function summarize_loop2(r)
    maxval = realmin(eltype(r))
    minval = realmax(eltype(r))
    s = 0
    for i = 1:length(r)
        d = r[i]
        s+=d
        if d > maxval
            maxval = d
        elseif d < minval
            minval = d
        end
    end
    s/length(r), maxval, minval
end
```
What’s the catch?

```python
function summarize_loop(r)
    maxval = realmin(eltype(r))
    minval = realmax(eltype(r))
    s = zero(eltype(r))
    for i = 1:length(r)
        d = r[i]
        s+=d
        if d > maxval
            maxval = d
        elseif d < minval
            minval = d
        end
    end
    s/length(r), maxval, minval
end
```

```julia
function summarize_loop2(r)
    maxval = realmin(eltype(r))
    minval = realmax(eltype(r))
    s = 0
    for i = 1:length(r)
        d = r[i]
        s+=d
        if d > maxval
            maxval = d
        elseif d < minval
            minval = d
        end
    end
    s/length(r), maxval, minval
end
```

<table>
<thead>
<tr>
<th></th>
<th>python</th>
<th>julia</th>
</tr>
</thead>
<tbody>
<tr>
<td>vectorized</td>
<td>54 ms</td>
<td>54 ms</td>
</tr>
<tr>
<td>loop</td>
<td>7400 ms</td>
<td>34 1200 ms</td>
</tr>
</tbody>
</table>
What’s the catch?

```
function summarize_loop(r)
    maxval = realmin(eltype(r))
    minval = realmax(eltype(r))
    s = zero(eltype(r))
    for i = 1:length(r)
        d = r[i]
        s+=d
        if d > maxval
            maxval = d
        elseif d < minval
            minval = d
        end
    end
    s/length(r), maxval, minval
end
```

```
function summarize_loop2(r)
    maxval = realmin(eltype(r))
    minval = realmax(eltype(r))
    s = 0
    for i = 1:length(r)
        d = r[i]
        s+=d
        if d > maxval
            maxval = d
        elseif d < minval
            minval = d
        end
    end
    s/length(r), maxval, minval
end
```

<table>
<thead>
<tr>
<th></th>
<th>python</th>
<th>julia</th>
</tr>
</thead>
<tbody>
<tr>
<td>vectorized</td>
<td>54 ms</td>
<td>54 ms</td>
</tr>
<tr>
<td>loop</td>
<td>7400 ms</td>
<td>34 1200 ms</td>
</tr>
</tbody>
</table>
Julia installation is straightforward, whether using precompiled binaries or compiling from source. Download and install Julia by following the instructions at http://julialang.org/downloads/.

The easiest way to learn and experiment with Julia is by starting an interactive session (also known as a read-eval-print loop or “repl”):

```
$ julia

   _   _   _   _   _   _   _   _   _   _   _   _   
  | | | | | | | | | | | | | | | | | | | | | | | | |
  | | | | | | | | | | | | | | | | | | | | | | | | |
  | | | | | | | | | | | | | | | | | | | | | | | | |
  | | | | | | | | | | | | | | | | | | | | | | | | |
  | | | | | | | | | | | | | | | | | | | | | | | | |
  | | | | | | | | | | | | | | | | | | | | | | | | |
  | | | | | | | | | | | | | | | | | | | | | | | | |

julia> 1 + 2
3

julia> ans
3
```

To exit the interactive session, type `^D` — the control key together with the `d` key or type `quit()`.

When run in interactive mode, `julia` displays a banner and prompts the user for input. Once the user has entered a complete expression, such as `1 + 2`, and hits enter, the interactive session evaluates the expression and shows its value. If an expression is entered into an interactive session with a trailing semicolon, its value is not shown. The variable `ans` is bound to the value of the last evaluated expression, whether it is shown or not. The `ans` variable is global in interactive sessions.
More info?

[julia language users - Google Groups](https://groups.google.com/d/forum/julia-users)

In order to keep the list spam free, your first message to the list is moderated, and may take some time to approve. Subsequent posts will not require moderation.

Julia Community

The julia-users mailing list is for discussion around the usage of Julia. ... The Julia Language SubReddit is a collection of various blog posts and articles related ...

The Julia Language

Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing ...
The First Rule of Program Optimization: Don't do it.
The Second Rule of Program Optimization (for experts only!): Don't do it yet.
-Michael A. Jackson
goto IJulia Demo link