in Astronomy: Better Together

Kyle Barbary
UC Berkeley / BIDS · github.com/kbarbary · @kylebarbary
“A fresh approach to technical computing”
I'm not going to tell you to stop using Python.
But...

• Some things are impossible to “vectorize,” or just very awkward.
• Possible to drop down to C but not easy
• As library writers, having to write “array-oriented” APIs can be constraining.

```python
class Points(object):
    """A container for two arrays giving x and y coordinates."""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __getattr__(self, i):
        return (self.x[i], self.y[i])
```

# ... other methods that operate element-wise
The “two language” problem

High-level dynamic language for rapid development (e.g., Python, R, Matlab, ...)

Interface code

Compiled language for performance-sensitive code (e.g., C, C++, Fortran)
## Julia is fast...

### Microbenchmarks

<table>
<thead>
<tr>
<th></th>
<th>Fortran</th>
<th>Julia</th>
<th>Python</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gcc 5.1.1</td>
<td>0.4.0</td>
<td>3.4.3</td>
<td>3.2.2</td>
</tr>
<tr>
<td>fib</td>
<td>0.70</td>
<td>2.11</td>
<td>77.76</td>
<td>533.52</td>
</tr>
<tr>
<td>parse_int</td>
<td>5.05</td>
<td>1.45</td>
<td>17.02</td>
<td>45.73</td>
</tr>
<tr>
<td>quicksort</td>
<td>1.31</td>
<td>1.15</td>
<td>32.89</td>
<td>264.54</td>
</tr>
<tr>
<td>mandel</td>
<td>0.81</td>
<td>0.79</td>
<td>15.32</td>
<td>53.16</td>
</tr>
<tr>
<td>pi_sum</td>
<td>1.00</td>
<td>1.00</td>
<td>21.99</td>
<td>9.56</td>
</tr>
<tr>
<td>rand_mat_stat</td>
<td>1.45</td>
<td>1.66</td>
<td>17.93</td>
<td>14.56</td>
</tr>
<tr>
<td>rand_mat_mul</td>
<td>3.48</td>
<td>1.02</td>
<td>1.14</td>
<td>1.57</td>
</tr>
</tbody>
</table>

Times relative to C (lower is better)
... but feels high-level.

```plaintext
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
```
... even at a very low level.

```plaintext
immutable Complex{T<:Real} <: Number
  re::T
  im::T
end

real(z::Complex) = z.re
imag(z::Complex) = z.im

+(z::Complex, w::Complex) = Complex(real(z) + real(w), imag(z) + imag(w))

*(z::Complex, w::Complex) = Complex(real(z) * real(w) - imag(z) * imag(w),
  real(z) * imag(w) + imag(z) * real(w))
```
A few more enticements:

- Call C and Fortran directly with zero overhead, no wrappers
  
  ```
  y = ccall((:sin, :libm), Float64, (Float64,), x)
  ```

- Zero-overhead quantities are possible:
  
  ```
  julia> 1V + 2V
  3 kg m^2s^{-3}A^{-1}
  ```

- Automatic differentiation is possible:
  
  ```
  f(x) = \sin(x) + x^2 + 4. * x^3
  
  gradient(f, 1.0)
  ```
A few more enticements:

- Inspect native code:
- Multithreading on its way (currently off by default)
Julia & Python: IJulia

Files

Running

Clusters

Select items to perform actions on them.

- 00 - Start Tutorial.ipynb
- Basics.ipynb
- Calling C and Python.ipynb
- Interactive Widgets.ipynb
- Metaprogramming.ipynb
- Multiple Dispatch.ipynb
- Plotting in Julia.ipynb
Julia & Python: PyCall

```julia
julia> using PyCall

julia> @pyimport numpy as np

julia> x = [-100, 39, 59, 55, 20]

julia> np.cumsum(x)
5-element Array{Int64,1}:
   -100
   -61
   -2
   53
   73

julia> @pyimport scipy.optimize as so

julia> function f(x)
    println("calling f($x)")
    cos(x) - x
end

julia> so.newton(f, 1.2);
calling f(1.2002199999999998)
calling f(0.7664554749111869)
calling f(0.7412167885608414)
calling f(0.7390978176492645)
calling f(0.7390851391787693)
0.7390851332151773
```
Julia Astro
Community Astronomy and Astrophysics packages for Julia

Packages

Packages are separated by functionality and Julia's declarative package manager takes care of resolving dependencies. You get just the functionality you need, and smaller packages lead to more rapid development.

Cosmology
Distances in the Universe
- ΛCDM and w0-wa cosmologies
- Open, closed, flat variants

DustExtinction
Dust extinction laws & maps
- CCM (1989) and O'Donnell (1994) dust laws
- SFD (1998) galactic dust map

ERFA
Low-level ERFA wrapper
- Wrapper for erfa C library
- Time system conversions

FITSIO
Reading and writing FITS files
- Image and table extensions
- Based on cfitsio C library

WCS
World Coordinate System transformations
- Wrapper for wcslib C library

wrap C libraries used in astropy
Other “registered” packages

helgee/JPLEphemeris
mweastwood/CasaCore
emmt/OIFITS
kbarbary/SkyCoords
jzuhone/YT
Example: SkyCoords

```julia
julia> using SkyCoords

julia> c1 = ICRSCoords(0.0, 0.0);

julia> convert(GalCoords, c1)
SkyCoords.GalCoords(1.6814027872278692, -1.0504884034813007)

julia> c1 = [ICRSCoords(0., 0.) for i=1:3]
3-element Array{SkyCoords.ICRSCoords,1}:
  SkyCoords.ICRSCoords(0.0, 0.0)
  SkyCoords.ICRSCoords(0.0, 0.0)
  SkyCoords.ICRSCoords(0.0, 0.0)

julia> sizeof(c1)
48
```
Example: SkyCoords
Rewriting all that Python?
Typing is not the hard part.
The hard part is figuring out what the code is supposed to do.

fk5 to fk4

<table>
<thead>
<tr>
<th>Tool 1</th>
<th>Tool 2</th>
<th>System 1</th>
<th>System 2</th>
<th>Median</th>
<th>Mean</th>
<th>Max</th>
<th>Std. Dev.</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>astropy</td>
<td>kapteyn</td>
<td>fk5</td>
<td>fk4</td>
<td>0.000004</td>
<td>0.000004</td>
<td>0.000006</td>
<td>0.000001</td>
<td>Plot</td>
</tr>
<tr>
<td>astropy</td>
<td>pyast</td>
<td>fk5</td>
<td>fk4</td>
<td>0.004450</td>
<td>0.004217</td>
<td>0.006194</td>
<td>0.001322</td>
<td>Plot</td>
</tr>
<tr>
<td>astropy</td>
<td>pyephem</td>
<td>fk5</td>
<td>fk4</td>
<td>0.564713</td>
<td>0.574624</td>
<td>1.020036</td>
<td>0.254973</td>
<td>Plot</td>
</tr>
<tr>
<td>astropy</td>
<td>pyslalib</td>
<td>fk5</td>
<td>fk4</td>
<td>0.004450</td>
<td>0.004217</td>
<td>0.006194</td>
<td>0.001322</td>
<td>Plot</td>
</tr>
<tr>
<td>kapteyn</td>
<td>pyast</td>
<td>fk5</td>
<td>fk4</td>
<td>0.004450</td>
<td>0.004216</td>
<td>0.006194</td>
<td>0.001322</td>
<td>Plot</td>
</tr>
<tr>
<td>kapteyn</td>
<td>pyephem</td>
<td>fk5</td>
<td>fk4</td>
<td>0.564712</td>
<td>0.574623</td>
<td>1.020032</td>
<td>0.254971</td>
<td>Plot</td>
</tr>
<tr>
<td>kapteyn</td>
<td>pyslalib</td>
<td>fk5</td>
<td>fk4</td>
<td>0.004449</td>
<td>0.004216</td>
<td>0.006194</td>
<td>0.001322</td>
<td>Plot</td>
</tr>
<tr>
<td>pyast</td>
<td>pyephem</td>
<td>fk5</td>
<td>fk4</td>
<td>0.564797</td>
<td>0.574896</td>
<td>1.019420</td>
<td>0.254471</td>
<td>Plot</td>
</tr>
<tr>
<td>pyast</td>
<td>pyslalib</td>
<td>fk5</td>
<td>fk4</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000003</td>
<td>0.000000</td>
<td>Plot</td>
</tr>
<tr>
<td>pyephem</td>
<td>pyslalib</td>
<td>fk5</td>
<td>fk4</td>
<td>0.564797</td>
<td>0.574896</td>
<td>1.019420</td>
<td>0.254471</td>
<td>Plot</td>
</tr>
<tr>
<td>astrolib</td>
<td>astropy</td>
<td>fk5</td>
<td>fk4</td>
<td>0.004450</td>
<td>0.004217</td>
<td>0.006189</td>
<td>0.001322</td>
<td>Plot</td>
</tr>
<tr>
<td>astrolib</td>
<td>kapteyn</td>
<td>fk5</td>
<td>fk4</td>
<td>0.004450</td>
<td>0.004217</td>
<td>0.006189</td>
<td>0.001322</td>
<td>Plot</td>
</tr>
<tr>
<td>astrolib</td>
<td>pyast</td>
<td>fk5</td>
<td>fk4</td>
<td>0.000007</td>
<td>0.000007</td>
<td>0.000014</td>
<td>0.000003</td>
<td>Plot</td>
</tr>
<tr>
<td>astrolib</td>
<td>pyephem</td>
<td>fk5</td>
<td>fk4</td>
<td>0.564799</td>
<td>0.574899</td>
<td>1.019430</td>
<td>0.254471</td>
<td>Plot</td>
</tr>
<tr>
<td>astrolib</td>
<td>pyslalib</td>
<td>fk5</td>
<td>fk4</td>
<td>0.000007</td>
<td>0.000007</td>
<td>0.000014</td>
<td>0.000003</td>
<td>Plot</td>
</tr>
</tbody>
</table>
My takeaways from Julia: C
My takeaways from Julia: C

Python

Cython

C

Python

Cython

Julia

PyPy + CFFI

C library
Takeaways from Julia: Modules

Various Julia projects are hosted under the following umbrella organizations on GitHub:

- JuliaLang – The language
- JuliaStats – Statistics
- JuliaOpt – Optimization
- JuliaParallel – Parallel programming in Julia
- JuliaDB – Various database drivers for Julia
- JuliaGPU – GPU computing
- BioJulia – Biology
- JuliaQuantum – Julia libraries for quantum science and technology
- JuliaAstro – Astronomy
- JuliaQuant – Finance
- JuliaSparse – Sparse matrix solvers
- JuliaDiff – Differentiation tools
- JuliaWeb – Web stack
- JuliaCI – Continuous Integration Support for Julia packages
- JuliaDSP – Digital signal processing
- JuliaGraphs – Graph Theory and Implementation
- JuliaLangEs – Julia resources in the Spanish language
Takeaways from Julia: Modules

“SciPy… really was a distribution masquerading as a library.
“... I made a mistake with SciPy because there was not good packaging in Python at the time.”

– Travis Oliphant
Thanks!
Sample Questions:

- What's bad about Julia?
- How can Python benefit from Julia?
- Is anyone actually using Julia for astronomy?

GitHub: @kbarbary
Twitter: @kylebarbary

Try it: juliabox.org
More talking: julialang.org/learning