Julia

...

Grant Matejka

What is Julia?

- Created in 2012, still very young but gaining fans
- Created by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman
- Wanted a high level and fast language

- On creation of Julia in regards to Matlab, Lisp, Python, Ruby, etc
  "We love all of these languages; they are wonderful and powerful. For the work we do — scientific computing, machine learning, data mining, large-scale linear algebra, distributed and parallel computing — each one is perfect for some aspects of the work and terrible for others. Each one is a trade-off.

  We are greedy: we want more." Why we created julia

- Written in the productivity language Julia, the Celeste project—which aims to catalogue all of the telescope data for the stars and galaxies in in the visible universe—**demonstrated the first Julia application to exceed 1 PF/s of double-precision floating-point performance** (specifically 1.54 PF/s). (2017 source)
  - AKA - 178 terabytes in 15 minutes

Why Julia?

- Claims to solve *“two language”* problem
- Big selling point is accessible and fast
- SPEED - ‘C like speed’
  - Thanks to JIT compilation
- Provides Lisp levels of generality (aka good macros)
- Speed and expressiveness allows Julia to be written in Julia
- Specifically made for scientific programming

How far did I get?

- Finished (?) interpreter with prebuilt AST
- Got some basic parsing in but was in general a little too new to fully incorporate it

  (~225 lines with test cases)

Code

```
@test interp(AppC(IdC('+'),
                  [NumC(2), NumC(2)]),
                  top_env)
      == NumV(4)
```

Code..

```
@test interp(AppC(LamC(["subtrctr", 'x', 'y'],
                AppC(IdC("subtrctr"), [IdC('y'), IdC('x')])),
                  [LamC(['x', 'y'],
                      AppC(IdC('-'), [IdC('y'), IdC('x')])),
              NumC(3),
              NumC(2)]), top_env) == NumV(-1)
```

Code Again...

```
top_env = [
 Pair('t', BooleanV(true)),
 Pair('f', BooleanV(false)),
 Pair('+', PrimV(addition)),
 Pair('-', PrimV(subtraction)),
 Pair('*', PrimV(multiplication)),
 Pair('/', PrimV(division)),
 Pair("<=", PrimV(leq))
]
```

Code Again Again...

```
1   #import Pkg
2   #Pkg.add("Match")
3   using Test
4   using Match
5
6   abstract type ExprC end
7
8   struct NumC <: ExprC n::Int end
9   struct StringC <: ExprC s end
10  struct IdC <: ExprC symbol end
11  struct IfC <: ExprC
12      cond
13      t_case
14      f_case
15  end
16  struct LamC <: ExprC
17      params
18      body
19  end
20  struct AppC <: ExprC
21      func
22      args
23  end
24
```

```
24
25  abstract type Value end
26
27  struct NumV <: Value n::Int end
28  struct BooleanV <: Value b::Bool end
29  struct StringV <: Value s::String end
30  struct PrimV <: Value op end
31  struct ClosV <: Value
32      params
33      body::ExprC
34      env
35  end
```

Last Bit of Code

```
94  function interp(expr::ExprC, env::Array)::Value
95    @match expr begin
96      num::NumC => return NumV(num.n)
97      str::StringC => return StringV(str.s)
98      id::IdC => return lookup(id.symbol, env)
99      lam::LamC =>
100       return ClosV(lam.params, lam.body, env)
101     if_expr::IfC =>
102       let cond = interp(if_expr.cond, env)
103         if typeof(cond) == BooleanV
104           if cond.b
105             interp(if_expr.t_case, env)
106           else
107             interp(if_expr.f_case, env)
108           end
109         else
110           error("If condition not boolean")
111         end
112       end
113     app::AppC =>
114       let func = interp(app.func, env)
115         # had issues doing a nested match so this is a workaround
116         if typeof(func) == PrimV
117           return func.op(interp(app.args[1], env), interp(app.args[2], env))
118         elseif typeof(func) == ClosV
119           argvals = map(arg -> interp(arg, env) , app.args)
120           new_env = build_new_env(argvals, func.params, env)
121           interp(func.body, new_env)
122         else
123           error("Cannot apply")
124         end
125       end
126     _ => error("Interp error: invalid expression")
127   end
128 end
```

Language Values

- Functions are first class, they can be passed around as arguments to other functions, etc
- Utilizes structs, but refers to them as composite types
- No difference between object and non object values
- From the docs ‘Only values, not variables, have types – variables are simply names bound to values.’ source

Scope - Lexical Scoping

Construct	Scope type	Allowed within
`module`, `baremodule`	global	global
`struct`	local (soft)	global
`for`, `while`, `try`	local (soft)	global or local
`macro`	local (hard)	global
`let`, functions, comprehensions, generators	local (hard)	global or local

**Soft vs Hard just means if shadowing global variable is allowed or not

Types

- Type system is dynamic but you can specify types statically
  - This can help performance
- Fancy definition “dynamic, nominative and parametric”
- Primitive types include bool, char and all kinds of floats/ints
  - But you can also specify your own
  - Def: “A primitive type is a concrete type whose data consists of plain old bits”

Syntax

Clearly follows current high level language syntax trends of today

Also caters well to data science/processing target audience

Some Cool Things:

- Whitespace doesn’t matter
- Need to use ‘end’
- Type system syntax seems a bit messy as a new user
- Also arrays start at index 1

Memory Management

(source)

According to the co-creator

- “non-compacting, generational, mark-and-sweep, tracing collector”
- Also random medium article I found says 2-level generational but couldn’t confirm

Would I take a job?

If I enjoyed data science/big data more, then yes, but as of now, I don’t think so.

Julia was great but I don’t think I’m part of the target audience for most of its intended uses.
I will definitely remember it for any side projects.

Thank You