Intro to Julia Programming Language

Bucharest Julia Meetup

9th of October 2019

Why another language?

The two language problem (1)

Performance:

- Fortran, C
- Slower to write code
- Steeper learning curve

Productivity:

- Python, Matlab
- Faster to write code
- Easier to learn

-> Technical barrier between the (usually) two teams developing the prototype and respectively the production ready code

Why another language?

- **Julia** was created to solve the two language problem, by combining productivity and performance:
  - Fast prototyping
  - Profiling tools to easily improve performance
  - No need for two development teams (prototyping in one language and translating/optimizing in another language):
    - Cost savings
    - Time savings
    - Technical barrier is lowered/removed:
      - Everything is coded in the same language
      - Everyone has full access to the internals
      - Much easier to become a contributor

Key Features of Julia

- Just-ahead-of-time-compiled[2]
- Dynamic type system
- Performance similar to C (*for* loops are fast ☺)
- Multiple dispatch[3]
- Built-in package manager

Julia Basics

0. Environment: Juno, REPL, Package Manager
1. Basic Types
2. Control Flow
3. Functions
4. Debugging
5. Profiling/Benchmarking
6. Special characters

-> On to the tutorial…

Q&A

Bucharest Julia Meetup

9th of October 2019

Side notes

1. More on the two language problem and the development of Julia by Stefan Karpinski, one of the co-creators of Julia.
2. ScottPJones: *"Julia's JIT compilation is rather different than what is referred to as JIT compilation in other languages, such as Java or JavaScript, where the language is interpreted (which may be interpreting instructions from a virtual machine such as the JVM), and the run-time decides if some code is being hit frequently enough to warrant compilation to native code. Julia first compiles to an AST representation (also expanding macros, etc), performs type inference, etc. When a method is called with types that haven't been used before to call that method, that's when Julia does it's magic and compiles a version of that method specialized for those types, using LLVM to generate the final machine code (just like most C and C++ implementations these days, as well as Rust and others). That also means that it's rare for Julia to have to dynamically dispatch methods based on the type of the arguments, which is one of the things that can really slow down other languages with dynamic types."*
3. Julia manual: *"Multiple dispatch is particularly useful for mathematical code, where it makes little sense to artificially deem the operations to "belong" to one argument more than any of the others: does the addition operation in x + y belong to x any more than it does to y? The implementation of a mathematical operator generally depends on the types of all of its arguments. Even beyond mathematical operations, however, multiple dispatch ends up being a powerful and convenient paradigm for structuring and organizing programs."*