Getting Started with Julia

By Erik Stryshak

Setup:

Data Structures:

Hello World:

Tips:

Gotchas:

Arrays start at 1, but can be changed

Useful Example:

UNORGANIZED:

[Julia](http://julialang.org/) is a high-level, high-performance dynamic programming language for numerical computing. It provides a sophisticated compiler, [distributed parallel execution](http://docs.julialang.org/en/stable/manual/parallel-computing/), numerical accuracy, and an [extensive mathematical function library](https://docs.julialang.org/en/latest/#Standard-Library-1). Julia’s Base library, largely written in Julia itself, also integrates mature, best-of-breed open source C and Fortran libraries for [linear algebra](http://docs.julialang.org/en/stable/stdlib/linalg/), [random number generation](http://docs.julialang.org/en/stable/stdlib/numbers/#random-numbers), [signal processing](http://docs.julialang.org/en/stable/stdlib/math/#signal-processing), and [string processing](http://docs.julialang.org/en/stable/stdlib/strings/#strings).

The Julia programming language fills this role: it is a flexible dynamic language, appropriate for scientific and numerical computing, with performance comparable to traditional statically-typed languages.

Because Julia's compiler is different from the interpreters used for languages like Python or R, you may find that Julia's performance is unintuitive at first. If you find that something is slow, we highly recommend reading through the [Performance Tips](https://docs.julialang.org/en/stable/manual/performance-tips/#man-performance-tips-1) section before trying anything else. Once you understand how Julia works, it's easy to write code that's nearly as fast as C.

Julia features optional typing, multiple dispatch, and good performance, achieved using type inference and [just-in-time (JIT) compilation](https://en.wikipedia.org/wiki/Just-in-time_compilation), implemented using [LLVM](https://en.wikipedia.org/wiki/Low_Level_Virtual_Machine). It is multi-paradigm, combining features of imperative, functional, and object-oriented programming.

Julia builds upon the lineage of mathematical programming languages, but also borrows much from popular dynamic languages, including [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)), [Perl](https://en.wikipedia.org/wiki/Perl_(programming_language)), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Lua](https://en.wikipedia.org/wiki/Lua_(programming_language)), and [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)).

The most significant departures of Julia from typical dynamic languages are:

* The core language imposes very little; the standard library is written in Julia itself, including primitive operations like integer arithmetic
* A rich language of types for constructing and describing objects, that can also optionally be used to make type declarations
* The ability to define function behavior across many combinations of argument types via [multiple dispatch](https://en.wikipedia.org/wiki/Multiple_dispatch)
* Automatic generation of efficient, specialized code for different argument types
* Good performance, approaching that of statically-compiled languages like C

User-defined types are as fast and compact as built-ins

Lightweight "green" threading ([coroutines](https://en.wikipedia.org/wiki/Coroutine))

Call C functions directly (no wrappers or special APIs needed)

Lisp-like macros and other metaprogramming facilities

Variables

* Variable names must begin with a letter (A-Z or a-z), underscore, or a subset of Unicode code points greater than 00A0
* The only explicitly disallowed names for variables are the names of built-in statements:

Style:

* Names of variables are in lower case.
* Word separation can be indicated by underscores ('\_'), but use of underscores is discouraged unless the name would be hard to read otherwise.
* Names of Types and Modules begin with a capital letter and word separation is shown with upper camel case instead of underscores.
* Names of functions and macros are in lower case, without underscores.
* Functions that write to their arguments have names that end in !. These are sometimes called "mutating" or "in-place" functions because they are intended to produce changes in their arguments after the function is called, not just return a value.

Compare Matlab Python w/ Numpy and Julia

https://cheatsheets.quantecon.org