

# Julia Quick Syntax Reference

A Pocket Guide for Data Science  
Programming

---

Antonello Lobianco



Apress®

# **Julia Quick Syntax Reference**

**A Pocket Guide for Data  
Science Programming**

**Antonello Lobianco**

**Apress®**

# *Julia Quick Syntax Reference: A Pocket Guide for Data Science*

## **Programming**

Antonello Lobianco  
Nancy, France

ISBN-13 (pbk): 978-1-4842-5189-8  
<https://doi.org/10.1007/978-1-4842-5190-4>

ISBN-13 (electronic): 978-1-4842-5190-4

Copyright © 2019 by Antonello Lobianco

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Steve Anglin  
Development Editor: Matthew Moodie  
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image designed by Freepik ([www.freepik.com](http://www.freepik.com))

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [editorial@apress.com](mailto:editorial@apress.com); for reprint, paperback, or audio rights, please email [bookpermissions@springernature.com](mailto:bookpermissions@springernature.com).

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/9781484251898](http://www.apress.com/9781484251898). For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

# Table of Contents

**About the Author .....ix**

**About the Technical Reviewer .....xi**

**Acknowledgments .....xiii**

**Introduction .....xv**

**Part I: Language Core .....1**

**Chapter 1: Getting Started .....3**

    1.1 Why Julia?.....3

    1.2 Installing Julia.....5

    1.3 Running Julia .....8

    1.4 Miscellaneous Syntax Elements .....10

    1.5 Packages.....11

        1.5.1 Using the Package Manager.....12

        1.5.2 Using Packages .....13

    1.6 Help System.....15

**Chapter 2: Data Types and Structures .....17**

    2.1 Simple Types (Non-Containers).....18

        2.1.1 Basic Mathematic Operations .....19

        2.1.2 Strings .....19

    2.2 Arrays (Lists) .....21

        2.2.1 Multidimensional and Nested Arrays.....25

    2.3 Tuples.....30

## TABLE OF CONTENTS

2.4 Named Tuples .....	31
2.5 Dictionaries .....	32
2.6 Sets .....	34
2.7 Memory and Copy Issues .....	34
2.8 Various Notes on Data Types .....	38
2.8.1 Random Numbers .....	39
2.8.2 Missing, Nothing, and NaN .....	39
<b>Chapter 3: Control Flow and Functions .....</b>	<b>41</b>
3.1 Code Block Structure and Variable Scope .....	41
3.2 Repeated Iteration: for and while Loops, List Comprehension, Maps .....	43
3.3 Conditional Statements: if Blocks, Ternary Operator .....	44
3.4 Functions .....	45
3.4.1 Arguments .....	47
3.4.2 Return Value .....	49
3.4.3 Multiple-Dispatch (aka Polymorphism) .....	49
3.4.4 Templates (Type Parameterization) .....	50
3.4.5 Functions as Objects .....	51
3.4.6 Call by Reference/Call by Value .....	51
3.4.7 Anonymous Functions (aka “Lambda” Functions) .....	52
3.4.8 Broadcasting Functions .....	53
3.5 Do Blocks .....	54
3.6 Exiting Julia .....	54
<b>Chapter 4: Custom Types .....</b>	<b>57</b>
4.1 Primitive Type Definition .....	58
4.2 Structure Definition .....	59
4.3 Object Initialization and Usage .....	60

4.4 Abstract Types and Inheritance .....	61
4.4.1 Implementation of the Object-Oriented Paradigm in Julia .....	63
4.5 Some Useful Functions Related to Types .....	66
<b>Chapter 5: Input/Output.....</b>	<b>67</b>
5.1 Reading (Input).....	68
5.1.1 Reading from the Terminal .....	68
5.1.2 Reading from a File .....	69
5.1.3 Importing Data from Excel.....	71
5.1.4 Importing Data from JSON.....	72
5.1.5 Accessing Web Resources (HTTP) .....	74
5.2 Writing (Output).....	75
5.2.1 Writing to the Terminal .....	75
5.2.2 Writing to a File .....	76
5.2.3 Exporting to CSV.....	77
5.2.4 Exporting Data to Excel .....	78
5.2.5 Exporting Data to JSON .....	79
5.3 Other Specialized IO.....	80
<b>Chapter 6: Metaprogramming and Macros.....</b>	<b>81</b>
6.1 Symbols .....	82
6.2 Expressions.....	83
6.2.1 Creating Expressions.....	84
6.2.2 Evaluating Symbols and Expressions .....	85
6.3 Macros .....	87
6.3.1 Macro Definition .....	87
6.3.2 Macro Invocation.....	88
6.3.3 String Macros .....	89

TABLE OF CONTENTS

<b>Chapter 7: Interfacing Julia with Other Languages.....</b>	<b>91</b>
7.1 Julia $\rightleftharpoons$ C .....	92
7.2 Julia $\rightleftharpoons$ C++ .....	94
7.2.1 Interactive C++ Prompt.....	94
7.2.2 Embed C++ Code in a Julia Program .....	95
7.2.3 Load a C++ Library .....	98
7.3 Julia $\rightleftharpoons$ Python .....	100
7.3.1 Embed Python Code in a Julia Program .....	101
7.3.2 Use Python Libraries.....	102
7.3.3 PyJulia: Using Julia in Python.....	103
7.4 Julia $\rightleftharpoons$ R .....	106
7.4.1 Interactive R Prompt.....	107
7.4.2 Embed R Code in a Julia Program .....	107
7.4.3 Use R Libraries .....	108
7.4.4 JuliaCall: Using Julia in R .....	109
<b>Chapter 8: Effectively Write Efficient Code .....</b>	<b>113</b>
8.1 Performance .....	114
8.1.1 Benchmarking .....	114
8.1.2 Profiling .....	116
8.1.3 Type Stability .....	120
8.1.4 Other Tips to Improve Performance.....	122
8.2 Code Parallelization .....	124
8.2.1 Adding and Removing Processes .....	124
8.2.2 Running Heavy Computations on a List of Items .....	126
8.2.3 Aggregate Results .....	127

8.3 Debugging.....	128
8.3.1 Introspection Tools.....	129
8.3.2 Debugging Tools .....	130
8.4 Managing Runtime Errors (Exceptions).....	133
<b>Part II: Packages Ecosystem .....</b>	<b>135</b>
<b>Chapter 9: Working with Data .....</b>	<b>137</b>
9.1 Using the DataFrames Package .....	138
9.1.1 Installing and Importing the Library .....	138
9.1.2 Creating a DataFrame or Loading Data .....	139
9.1.3 Getting Insights About the Data .....	141
9.1.4 Filtering Data (Selecting or Querying Data) .....	142
9.1.5 Editing Data .....	145
9.1.6 Editing Structure.....	146
9.1.7 Managing Missing Values .....	148
9.1.8 The Split-Apply-Combine Strategy .....	149
9.1.9 Pivoting Data .....	153
9.1.10 Dataframe Export.....	159
9.2 Using IndexedTables .....	161
9.2.1 Creating an IndexedTable (NDSparse) .....	162
9.2.2 Row Filtering .....	163
9.2.3 Editing/Adding Values.....	164
9.3 Using the Pipe Operator .....	164
9.4 Plotting.....	166
9.4.1 Installation and Backends .....	166
9.4.2 The Plot Function.....	169
9.4.3 Plotting from DataFrames.....	172
9.4.4 Saving.....	175



TABLE OF CONTENTS

<b>Chapter 10: Mathematical Libraries .....</b>	<b>177</b>
10.1 JuMP, an Optimization Framework.....	178
10.1.1 The Transport Problem: A Linear Problem.....	180
10.1.2 Choosing Between Pizzas and Sandwiches, a Non-Linear Problem .....	188
10.2 SymPy, a CAS System .....	191
10.2.1 Loading the Library and Declaring Symbols.....	191
10.2.2 Creating and Manipulating Expressions .....	192
10.2.3 Solving a System of Equations .....	193
10.2.4 Retrieving Numerical Values.....	193
10.3 LsqFit, a Data Fit Library .....	194
10.3.1 Loading the Libraries and Defining the Model.....	195
10.3.2 Parameters .....	195
10.3.3 Fitting the Model .....	196
10.3.4 Retrieving the Parameters and Comparing them with the Observations.....	196
<b>Chapter 11: Utilities .....</b>	<b>199</b>
11.1 Weave for Dynamic Documents .....	199
11.2 Zip Files.....	203
11.2.1 Writing a Zip Archive.....	203
11.2.2 Reading from a Zipped Archive .....	204
11.3 Interact and Mux: Expose Interacting Models on the Web .....	206
11.3.1 Importing the Libraries .....	206
11.3.2 Defining the Logic of the Model.....	206
11.3.3 Defining Controls and Layout .....	207
11.3.4 Providing Widgets to Web Users.....	208
<b>Index.....</b>	<b>211</b>

# About the Author

**Antonello Lobianco**, PhD is a research engineer employed by a French Grande école (Polytechnic University). He works on biophysical and economic modeling of the forest sector and is responsible for the Lab Models portfolio. He uses C++, Perl, PHP, Python, and Julia. He teaches environmental and forest economics at the undergraduate and graduate levels and modeling at the PhD level. He has been following the development of Julia as it fits his modeling needs, and he is the author of several Julia packages (search for *sylvaticus* on GitHub for more information).

# About the Technical Reviewer

**Germán González-Morris** is a polyglot software architect/engineer with 20+ years in the field. He has knowledge of Java (EE), Spring, Haskell, C, Python, and JavaScript, among others. He works with web distributed applications. Germán loves math puzzles (including reading Knuth) and swimming. He has tech reviewed several books, including an application container book (Weblogic), as well as titles covering various programming languages (Haskell, TypeScript, WebAssembly, Math for Coders, and RegExp, for example). You can find more information on his blog (<https://devwebcl.blogspot.com/>) or Twitter account (@devwebcl).

# Acknowledgments

This work has been supported by the French National Research Agency through the Laboratory of Excellence, ARBRE, part of the “Investissements d’Avenir” program (ANR 11 – LABX-0002-01).

I want to thank Germán González-Morris for his valuable help in finding errors in the code and improving the description of the language. I want to also thank Mark Powers, the Apress coordinating editor, for his numerous “check ins” that pushed me to continue and finish the book.

This has been possible thanks to the understanding and support of my family.

# Introduction

This Julia quick syntax reference book covers the main syntax elements of the Julia language as well as some of its more important packages.

The first chapter explains the basic skills needed to set up the software you need to run and develop Julia programs, including managing Julia packages.

Chapter 2 presents the many predefined types (integers, strings, arrays, etc.) and the methods to work with them. Memory and copy issues are also presented in this chapter, together with an important discussion about the implementation of the various concepts of *missingness*.

After the basic data types have been introduced, Chapter 3 deals with how to organize them in a sequence of logical statements to compose your program. Control flow, functions, blocks, and scope are all discussed in this chapter.

In Chapter 4, we extend our discussion to custom types—in Julia, both primitive and composite types can be custom-defined—and to their organization in the program, either using inheritance or composition. This chapter will be of particular use to readers accustomed to other object-oriented programs, in order to see how to apply object-oriented concepts in Julia.

Chapter 5 explains how to retrieve the inputs needed by your program from a given source (the terminal, a text/CSV/Excel/JSON file, or a remote resource) and conversely, to export the outputs of your program.

In Chapter 6, we discuss a peculiar feature of Julia, that is, the possibility to manipulate the code itself after it has been parsed, but before it is compiled and run. This paves the way to powerful macro programming. We discuss it and present the concepts of *symbols* and *expressions* in Chapter 6.

## INTRODUCTION

Julia is a relatively new language, and while the package ecosystem is developing extremely rapidly (as most packages can be written directly in the Julia language alone), it is highly likely that you will still need libraries for which a direct port to Julia is not yet available. Conversely, your main workflow may be in another, slower, high-level language and you may want to use Julia to implement some performant-critical tasks. Chapter 7 shows how to use C, C++, Python, and R code and their relative libraries in Julia and, conversely, embed Julia code in Python or R programs.

The following chapter (Chapter 8) gives a few recommendations for writing efficient code, with runtime performances comparable to compiled languages. We also deal here with *programmer's efficiency*, discussing profiling and debugging tools and with a short introduction to runtime exceptions.

This completes the discussion of the *core* of the language. Julia, however, has been designed as a thin language where most features are provided by external packages, either shipped with Julia itself (a sort of Julia Standard Library) or provided by third parties.

Therefore, the second part of the book deals with this Julia package ecosystem. Chapter 9 introduces the main packages for working with numerical data: storage with data structure packages like DataFrames and IndexedTables; munging with DataFramesMeta, Query, and Pipe; and visualization with the Plot package.

If Chapter 9 deals with processing numerical data, Chapter 10 deals with mathematical libraries for more theoretical work. JuMP is an acclaimed “algebraic modeling language” for numerical optimization (and can be in itself the primary reason to learn about Julia). We present two complete examples with linear and non-linear models. The second model is then rewritten to be analytically resolved with SymPy, which is a library for symbolic computation, e.g. the analytical resolution of derivatives, integrals, and equations (and systems of equations). Chapter 10 ends with a presentation of LsqFit, a powerful and versatile library to fit data. Finally, Chapter 11 concludes the book with a series of tools that are of

more general use, like composing dynamic documents with Wave, dealing with ZIP files with ZipFile, and exposing a given Julia model on the web with Interact and Mux. Examples given in the text are intentionally trivial. They are minimal working examples of the syntax for the concepts they explain. If you are looking for recipes directly applicable to your domain, a “cookbook” kind of book may be more convenient.

While each package has been tested specifically with Julia 1.2 and 1.3-rc4, thanks to the Julia developers’ commitment to a stable API, they should remain relevant for the entire 1.x series. Concerning third-party packages, we report the exact version we tested our code with. The section entitled “Using the Package Manager” in Chapter 1 explains how to update a package to a given version if subsequent versions of the package break the API.

In such cases, please report the problem to us using the form at <https://julia-book.com>. We will regularly publish updates and errata on this site, where a discussion forum focused on the book is also available.

# **PART I**

## **Language Core**



## CHAPTER 1

# Getting Started

## 1.1 Why Julia?

With so many programming languages available, why create yet another one? Why invest the time to learn Julia? Is it worth it?

One of the main arguments in favor of using Julia is that it contributes to improving a trade-off that has long existed in programming—fast coding versus fast execution.

On the one side, Julia allows the developer to code in a dynamic, high-level language similar to Python, R, or MATLAB, interacting with the code and having powerful expressivity (see Chapter 6, for example).

On the other side, with minimum effort, developers can write programs in Julia that run (almost) as fast as programs written in C or FORTRAN.

Wouldn't it be better, though, to optimize existing languages, with their large number of libraries and established ecosystems, rather than create a new language from scratch?

Well, yes and no. Attempts to improve runtime execution of dynamic languages are numerous. PyPy (<https://pypy.org>), Cython (<https://cython.org>), and Numba (<https://numba.pydata.org>) are three notable examples for the Python programming language. They all clash with one fact: Python (and, in general, all the current dynamic languages) was designed before the recent development of just-in-time (JIT) compilers, and hence it offers features that are not easy to optimize. The optimization tools either fail or require complex workarounds in order to work.

Conversely, Julia has been designed from the ground up to work with JIT compilers, and the language features—and their internal implementations—have been carefully considered in order to provide the programmer with the expected productivity of a modern language, all while respecting the constraints of the compiler. The result is that Julia-compliant code is guaranteed to work with the underlying JIT compiler, producing in the end highly optimized compiled code.

---

### *The Shadow Costs of Using a New Language*

If it is true that the main “costs” of using a new language relate to learning the language and having to abandon useful libraries and comfortable, feature-rich development editors that you are accustomed to, it is also true that in the Julia case these costs are mitigated by several factors:

- The language has been designed to syntactically resemble mainstream languages (you’ll see it in this book!). If you already know a programming language, chances are you will be at ease with the Julia syntax.
- Julia allows you to easily interface your code with all the major programming languages (see Chapter 7, “Interfacing Julia with Other Languages”), hence reusing their huge sets of libraries (when these are not already ported to Julia).

- The development environments that are available—e.g., Juno (<https://junolab.org>), IJulia Jupiter kernel (<https://github.com/JuliaLang/IJulia.jl>), and VSCode Julia plugin (<https://github.com/JuliaEditorSupport/julia-vscode>)—are frankly quite cool, with many common features already implemented. They allow you to be productive in Julia from the first time you code with it.
- 

Apart from the breakout in runtime performances from traditional high-level dynamic languages, the fact that Julia was created from scratch means it uses the best, most modern technologies, without concerns over maintaining compatibility with existing code or internal architectures. Some of the features of Julia that you will likely appreciate include built-in Git-based package manager, full code introspection, multiple dispatches, in-core high-level methods for parallel computing, and Unicode characters in variable names (e.g., Greek letters).

Thanks to its computational advantages, Julia has its natural roots in the domain of scientific, high-performance programming, but it is becoming more and more mature as a general purpose programming language. This is why this book does not focus specifically on the mathematical domain, but instead develops a broad set of simple, elementary examples that are accessible even to beginner programmers.

## 1.2 Installing Julia

Julia code can be run by installing the Julia binaries for your system available in the download section (<http://julialang.org/downloads/>) of the Julia Project website (<https://julialang.org>).

The binaries ship with a Julia interpreter console (aka, the “REPL”—Read, Eval, Print, Loop), where you can run Julia code in a command-line fashion.

For a better experience, check out an Integrated Development Environment, for example, Juno (<http://junolab.org/>) an IDE based on the Atom (<https://atom.io>) text editor, or IJulia (<https://github.com/JuliaLang/IJulia.jl>), the Julia Jupiter (<http://jupyter.org/>) backend.

Detailed setup instructions can be found on their respective sites, but in a nutshell, the steps are pretty straightforward.

- For Juno:
  - Install the main Julia binaries first.
  - Download, install, and open the Atom text editor (<https://atom.io>).
  - From within Atom, go to the Settings ► Install panel.
  - Type `uber-juno` into the search box and press Enter. Click the Install button on the package with the same name.
- For IJulia:
  - Install the main Julia binaries first.
  - Install the Python-based Jupyter Notebook server using the favorite tools of your OS (e.g., the Package Manager in Linux, the Python `pip` package manager, or the Anaconda distribution).
  - From a Julia console, type `using Pkg; Pkg.update(); Pkg.add("IJulia"); Pkg.build("IJulia")`.

- The IJulia kernel is now installed. Just start the notebook server and access it using a browser.

You can also choose, at least to start with, not to install Julia at all, and try instead one of the online computing environments that support Julia. For example, JuliaBox (<https://juliabox.com/>), CoCalc (<https://cocalc.com/doc/software-julia.html>), Nextjournal (<https://nextjournal.com>), and Binder (<https://mybinder.org>).

---

### *Some tricks for Juno and IJulia*

- Juno can:
    - Enable block selection mode with `ALT` + `SHIFT`.
    - Run a selection of code by selecting it and either selecting Run Block or typing `SHIFT` + `Enter` on Windows and Linux or `CMD` + `Enter` on Mac.
    - Comment/uncomment a block of code with `CTRL` + `/` (Windows and Linux) or `CMD` + `/` (Mac).
  - IJulia:
    - Check out the many keyboard shortcuts available from Help ► Keyboard Shortcuts.
    - Need to run Julia in a computational environment for a team or a class? Use JupyterHub (<https://github.com/jupyterhub/jupyterhub>), the multi-user solution based on Jupyter.
-

## 1.3 Running Julia

There are many ways to run Julia code, depending on your needs:

1. Julia can run interactively in a console. Start `julia` to obtain the REPL console, and then type the commands there (type `exit()` or use CTRL+D when you are finished).
2. Create a script, i.e. a text file ending in `.jl`, and let Julia parse and run it with `julia myscript.jl [arg1, arg2,...]`.

Script files can also be run from within the Julia console. Just type `include("myscript.jl")`.

3. In Linux or on MacOS, you can instead add at the top of the script the location of the Julia interpreter on your system, preceded by `#!` and followed by an empty row, e.g. `#!/usr/bin/julia` (You can find the full path of the Julia interpreter by typing `which julia` in a console.). Be sure that the file is executable (e.g., `chmod +x myscript.jl`).

You can then run the script with `./myscript.jl`.


4. Use an Integrated Development Environment (such as those mentioned), open a Julia script, and use the run command specific to the editor.

You can define a global (for all users of the computer) and local (for a single user) Julia file that will be executed at any startup, where you can for example define functions or variables that should always be available. The location of these two files is as follows:

- Global Julia startup file: [JULIA\_INSTALL\_FOLDER]\etc\julia\startup.jl (where JULIA\_INSTALL\_FOLDER is where Julia is installed)
- Local Julia startup file: [USER\_HOME\_FOLDER]\.julia\config\startup.jl (where USER\_HOME\_FOLDER is the home folder of the local user, e.g. %HOMEPATH% in Windows and ~ in Linux)

Remember to use the path with forward slashes ( / ) with Linux. Note that the local config folder may not exist. In that case, just create the config folder as a .julia subfolder and start the new startup.jl file there.

---

 Julia keeps all the objects created within the same work session in memory. You may sometimes want to free memory or “clean up” your session by deleting no longer needed objects. If you want to do this, just restart the Julia session (you may want to use the trick mentioned at the end of Chapter 3) or use the `Revise.jl` (<https://github.com/timholy/Revise.jl>) package for finer control.

You can determine which version of Julia you are using with the `versioninfo()` option (within a Julia session).

---