

Solving the n -language problem: A ecologist's guide to learning Julia

Michael D. Catchen^{1,2} [Timothée Poisot](#)^{3,2}

¹ McGill University ² Québec Centre for Biodiversity Sciences ³ Université de Montréal

Correspondance to:

Michael D. Catchen — michael.catchen@mcgill.ca

This work is released by its authors under a CC-BY 4.0 license



Last revision: *September 9, 2022*

Julia is a good language, ecologists should learn it.

1 Outline

- 2 • Why should ecologists learn julia?
 - 3 – Well, there are the criteria that are directly measureable that make it better than R/Python:
 - 4 * fast
 - 5 * native support on GPUs
 - 6 – But there are also criteria that are more subjective, and that take experience and practice using
 - 7 the language to appreciate
 - 8 * clever use of dispatch patterns
 - 9 * use of one-liners
 - 10 * using parameterized types well
 - 11 – You will learn how to be a better programmer in *any* language, because smart use of julia
 - 12 requires understanding some fundamental concepts in programming that are ‘hidden’ from
 - 13 users in R/python
 - 14 – The biggest reason *not* to use julia is that the ecology/evolution package ecosystem in R is
 - 15 larger, and the ML ecosystem in python is more popular. However:
 - 16 * you can call *any* R/python function/library using RCall/PyCall in julia
 - 17 * More packages isn’t necessarily better when they don’t work together

18

19 Abstract

20

21 Introduction

22 In order to measure, understand, and mitigate the consequences of anthropogenic change on ecosystems
23 the services they provide, ecologists need a set of computational tools (Urban *et al.* 2022). These tools must
24 be performant, but crucially modular and interfaceable (McIntire *et al.* 2022).

25 Ecological data is often difficult to access and reuse (Gonzalez & Peres-Neto 2015; Poisot *et al.* 2019).
26 Many sources of ecological, evolutionary, and environmental data exist, but synthesizing this data into a
27 single product suitable for analysis often remains tedious as data are not in formats that can be easily
28 combined or interfaced. Here we propose that we can solve this problem through standardization
29 (Zimmerman 2008)—developing a common definition such that data collected in a variety of contexts can
30 be assimilated while minimizing the overhead of data cleaning and wrangling.

31 A common representation of ecological data will have three primary benefits: it will **1)** enable new forms
32 of analysis by making it easier to combine data from different sources (Heberling *et al.* 2021), **2)** enable
33 continuous integration of new data for next-generation biodiversity monitoring (Kühl *et al.* 2020), and **3)**
34 aid in open sharing and reproducibility of published results (Zimmerman 2008; Borregaard & Hart 2016).
35 Here, we briefly review approaches to data standardization developed in other fields, in order to determine
36 what makes an open standard succeed in promoting data sharing, and what doesn't. Based on the
37 properties of good standards we identify, we propose building a living standard for ecological data in the
38 Julia programming language, and argue this is necessary to obtain the three primary benefits of
39 standardization mentioned earlier.

40 The so-called “two-language problem” in computational science, where it is easier for a researcher to
41 developer a prototype of a model/simulation in a high-level language, like Python or R, and later have to
42 port the model to a lower-level compiled language because the performance of these compiled languages
43 (e.g. C++/Fortran) is orders of magnitude faster than high level interpreted languages. In fact, many of the
44 most popular tools in higher-level languages are actually thin wrappers around a compiled (often C++)
45 base (e.g. tidyverse, keras, numpy, TensorFlow, scikit-learn, pandas, etc.). However, the skills required to
46 use or debug—let alone write—scientific software in these lower level languages is not often taught.

47 We propose that that Julia has certain properties absent in other popular languages for scientific
48 computing that make it particularly suited for the development of a cohesive, modular, and extendible set
49 of tools ideal for the development of a platform for ecological analysis [a].

50 **The nature of computation in Julia**

51 [Figure 1 about here.]

52 **Types**

53 Why is this useful for ecologists? Often times in ecology, the same information is represented in different
54 formats. Two packages in R might not agree on what the “correct” format to represent information is.

55 At the core of the Julia language is its *type system*. Type systems can often be alienating to those who
56 learned programming in so-called *dynamically* typed languages (like R, python, and JavaScript). In
57 dynamically-typed languages, `x = 5`, and `x = "hello world"` and the language won't care that you
58 changed the type of information that was stored in `x` from a number to a string. Practically, this form of
59 dynamic-typing was adopted because it is far more convenient to write code like that above than defining
60 variables with explicit types, e.g. how you would in C: `char c = "a";` and `int x = 5;`.

61 Julia doesn't require explicit type declarations, meaning `x = 5` is perfectly valid code, but internally Julia is
62 doing the bookkeeping of what type of information is stored in `x`, from an `Int64` to a `String` in the above
63 example.

64 Using explicit types is central to Julia's speed, but also enables much of its most unique and user-friendly
65 functionality, primarily the use of a *multiple-dispatch* system.

66 **Dispatch**

67 *Dispatch* refers to the way a computer program decides what function to call.

68 In many statically-typed languages, you are allow to use the same function name more than once.

69 **Doing computational science in Julia**

70 **Managing Data**

71 `DataFrames.jl` and `DFMeta`.

72 **Doing statistics and machine learning**

73 7. Learn about the statistics ecosystem: `StatsBase`, `Statistics`, `GLM`, `MLJ`, `Flux`, `Turing`

74 **Doing simulation**

- 75 8. Learn about the simulation libraries (DiffEq, DynamicGrids)
- 76 9. Learn how various statistics/simulation libraries work together

77 **Discussion**

78 Defining a living standard for ecological data in Julia will make it easier to combine data from different
79 sources by splitting the process of data aggregation from the process of analysis. Integrating data from a
80 particular study, or a new database, would be as simple as implementing the interface from the data
81 source to the standardized types. Data from individual studies could be incorporated into public
82 repositories containing both the raw data and the interface to Julia data structures, and this combined
83 data/interface package is all that is needed to either reproduce the results or incorporate that particular
84 study's data into analysis. This will make combining data from multiple sources easier, and yield benefits
85 for the development and implementation of novel methods, as the software for analysis becomes separate
86 from the software for data cleaning and aggregation.

87 We envision a modern set of tools for ecology in Julia based around the standardized types. Far outside of
88 ecology, the term “ecosystem” is used metaphorically to describe a set of software tools that work together.
89 We imagine multiple “trophic-levels” of packages for ecological science in Julia based around the “basal”
90 set of standardized types — a modular set of tools that can be chained together create arbitrarily complex
91 analysis pipelines. that can be scaled to meet the needs of next-generation biodiversity monitoring.

92 Borregaard, M.K. & Hart, E.M. (2016). Towards a more reproducible ecology. *Ecography*, 39, 349–353.

93 Gonzalez, A. & Peres-Neto, P.R. (2015). Act to staunch loss of research data. *Nature*, 520, 436–436.

94 Heberling, J.M., Miller, J.T., Noesgaard, D., Weingart, S.B. & Schigel, D. (2021). Data integration enables
95 global biodiversity synthesis. *Proceedings of the National Academy of Sciences*, 118.

96 Köhl, H.S., Bowler, D.E., Bösch, L., Bruehlheide, H., Dauber, J., Eichenberg, David., *et al.* (2020). Effective
97 Biodiversity Monitoring Needs a Culture of Integration. *One Earth*, 3, 462–474.

98 McIntire, E.J.B., Chubaty, A.M., Cumming, S.G., Andison, D., Barros, C., Boisvenue, C., *et al.* (2022).

99 PERFICT: A Re-imagined foundation for predictive ecology. *Ecology Letters*, 25, 1345–1351.

- 100 Poisot, T., Bruneau, A., Gonzalez, A., Gravel, D. & Peres-Neto, P. (2019). Ecological Data Should Not Be So
101 Hard to Find and Reuse. *Trends in Ecology & Evolution*, 34, 494–496.
- 102 Urban, M.C., Travis, J.M.J., Zurell, D., Thompson, P.L., Synes, N.W., Scarpa, A., *et al.* (2022). Coding for
103 Life: Designing a Platform for Projecting and Protecting Global Biodiversity. *BioScience*, 72, 91–104.
- 104 Zimmerman, A.S. (2008). New Knowledge from Old Data: The Role of Standards in the Sharing and
105 Reuse of Ecological Data. *Science, Technology, & Human Values*, 33, 631–652.

1 Defining the types

```
abstract type Pet end
struct Dog <: Pet
  name
end
struct Cat <: Pet
  name
end
```

2 Defining the methods

```
meet(🐶::Dog, 🐱::Cat) = "$(🐶.name) meets $(🐱.name) and barks"
meet(🐶::Dog, 🐾::Dog) = "$(🐶.name) meets $(🐾.name) and sniffs"
meet(🐱::Cat, 🐶::Dog) = "$(🐱.name) meets $(🐶.name) and hisses"
meet(🐱::Cat, 🐱::Cat) = "$(🐱.name) meets $(🐱.name) and slinks"
```

3 Creating instances of types

```
fido = Dog("Fido")
sparky = Dog("Sparky")

tabby = Cat("Tabby")
panko = Cat("Panko")
```

4 Calling the methods

```
meet(sparky, tabby)
> Sparky meets Tabby and barks
meet(fido, sparky)
> Fido meets Sparky and sniffs
meet(panko, fido)
> Panko meets Sparky and hisses
meet(tabby, panko)
> Tabby meets Panko and slinks
```

Figure 1: TODO: caption. Adapted from Karpinski 2019 “The unreasonable effectiveness of multiple dispatch”