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

Julia is a good language, ecologists should learn it.

Keywords:
computational ecology
scientific computing
machine learning
numerical ecology

1 Outline

- Why should ecologists learn julia?
    - Well, there are the criteria that are directly measureble that make it better than R/Python:
        - fast
        - native support on GPUs
    - But there are also criteria that are more subjective, and that take experience and practice using the language to appreciate
        - clever use of dispatch patterns
        - use of one-lienrs
        - using parameterized types well
    - You will learn how to be a better programmer in *any* language, because smart use of julia requires understanding some fundemantal concepts in programming that are 'hidden' from users in R/python
    - The biggest reason *not* to use julia is that the ecology/evolution package ecosystem in R is larger, and the ML ecosystem in python is more popular. However:
        - you can call *any* R/python function/library using RCall/PyCall in julia
        - More packages isn't necessarily better when they don't work together

2 Abstract

3

Introduction

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

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

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

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

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

4

The nature of computation in Julia

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

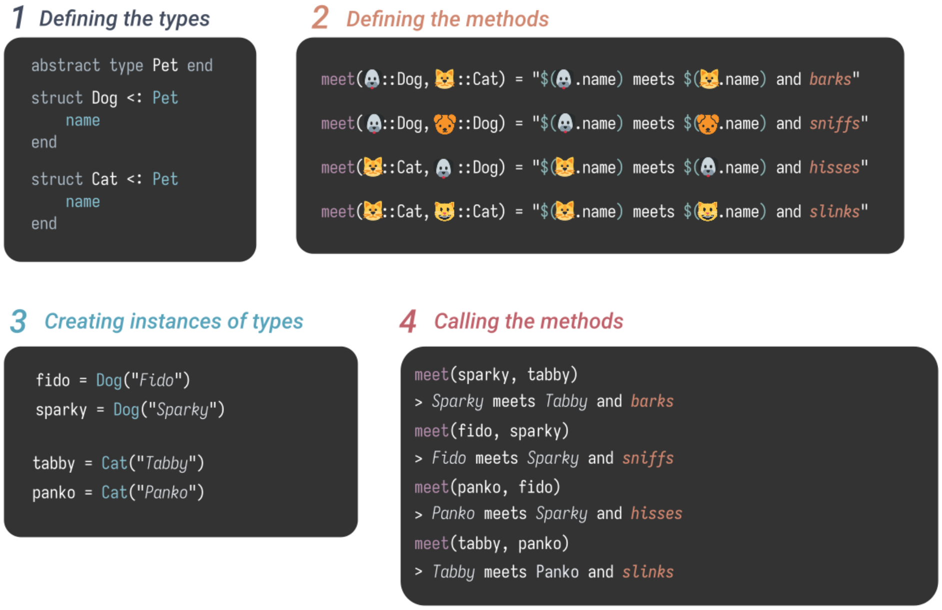
At the core of the Julia language is its *type system*. Type systems can often be alienating to those who learned programming in so-called *dynamically* typed languages (like R, python, and JavaScript). In dynamically-typed languages, `x= 5`, and `x = "hello world"` and the language won't care that you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

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

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

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

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



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”

5 Doing computational science in `Julia`

5.1. Managing Data DataFrames.jl and DFMeta.

5.2. Doing statistics and machine learning

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

5.3. Doing simulation

8. Learn about the simulation libraries (DiffEq, DynamicGrids)
9. Learn how various statistics/simulation libraries work togethe

6 Discussion

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

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

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

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

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

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

McIntire, E.J.B., Chubaty, A.M., Cumming, S.G., Andison, D., Barros, C., Boisvenue, C., *et al.* (2022). PERFICT: A Re-imagined foundation for predictive ecology. *Ecology Letters*, 25, 1345–1351.

Poisot, T., Bruneau, A., Gonzalez, A., Gravel, D. & Peres-Neto, P. (2019). Ecological Data Should Not Be So Hard to Find and Reuse. *Trends in Ecology & Evolution*, 34, 494–496.

Urban, M.C., Travis, J.M.J., Zurell, D., Thompson, P.L., Synes, N.W., Scarpa, A., *et al.* (2022). Coding for Life: Designing a Platform for Projecting and Protecting Global Biodiversity. *BioScience*, 72, 91–104.

Zimmerman, A.S. (2008). New Knowledge from Old Data: The Role of Standards in the Sharing and Reuse of Ecological Data. *Science, Technology, & Human Values*, 33, 631–652.