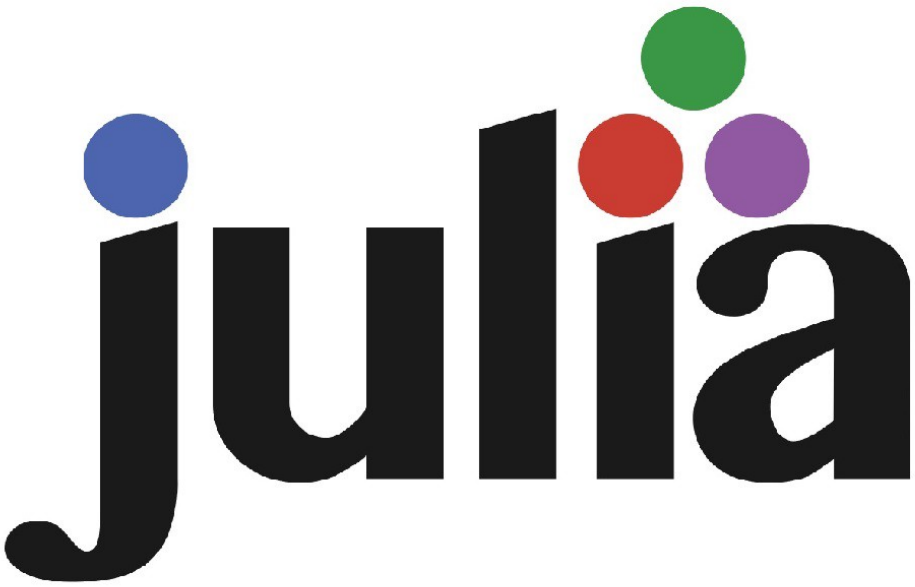
Introducing Julia: An Alternative to Python and R for Data Science

Meet the programing language of the 21st century, aimed for scientific computing, data science, and machine learning.

Dario Radečić
Jan 23 · 6 min read ★

When someone says to me that there's a new kid in the block, and in this situation kid = programming language, I get a bit skeptical. For a while now I've been a heavy Python user, looking at it for 8 hours at work, and an additional 3–4 hours at home — and I like it, there are no reasons not to.

Well, maybe there are a few…



As you can imagine, someone who is coding in Python for half or even more of the day can be a bit skeptical when it comes to trying out new languages. A while back a couple of folks tried to convince me to switch to R, and I honestly tried but disliked so much about it.

Data science isn't all about statistics, you also have to create APIs and stuff, so the end-user can somehow interact with what you've made, so for me, it's only natural to choose programming language instead of a statistical language. But you might disagree, and that's fine.

Anyway, let's get back on the topic. ***So what is Julia?*** According to a quick web search, Julia is a high-level, high-performance, dynamic, and general-purpose programming language created by MIT and is mostly used for numerical analysis. The work on the language started around 2009, and the first release was in 2012. So the language has been around 8 years now, not a short period, but surely not long as for example, Python has.

That's quite a sentence. You can think of it as a mixture of R and Python, but faster because it's a compiled language. Some of the major advantages Julia has over R and Python are:

1. **It's compiled** — Julia is just-in-time (JIT) compiled, and at it's best can approach or match the speed of C language
2. **It can call Python, C, and Fortran libraries** — yeah, you've read that correctly, and we'll explore this in more details some other time
3. **Parallelism** — parallelization in Julia is more refined than in Python

However, for every advantage there is a disadvantage, to name a few:

1. **Array indexing starts at 1** — that's just insane for me
2. **Not as mature as Python**
3. **Not as many packages as with Python** — but you can call Python libraries through Julia

Okay, I just hope you're still here, as now we'll explore some key concepts of Julia programming language. First, however, we should install the language, right?

. . .

How to Install Julia

In this quick section, we'll quickly go over the process of installing Julia, and also setting up a kernel for Jupyter. If you don't use Jupyter feel free to download plugins for your editor of choice.

Let's get started. Open up the following URL, and hit that big green button that says **Download**. You'll be asked to specify your OS, which is Windows x64 for me, and then you can download the file. The installation itself is straightforward, only press **Next** a couple of times and you'll be ready to go.

Jupyter-wise we still have some work to do. You'll need to open up the **Julia prompt**, the window should look like this:

Now enter the following two lines:

```
using Pkg
Pkg.add("IJulia")
```

After a minute or so the Julia kernel will be configured with Jupyter, and we can verify that by opening up the **JupyterLab**, where now I have the Julia interpreter alongside with the Python one:

That's great, now create a Julia Notebook and you're ready to have some fun!

. . .

Let's Explore the Basics

In your newly created Notebook you can define a variable just as you would do with Python:

By default, the variable value will get printed out, to avoid that you can put the semicolon (;) at the end. If you've noticed, the strings need to be surrounded with **double quotation marks**, as single quotation marks are reserved for character types.

You can easily **concatenate** two or more strings:

The multiplication sign might be strange at first, but you'll get used to it. You can also easily check the data type of any given variable (or value):

Julia is similar to Python for the **array** creation (well, to a degree), where you don't have to specify data type or the number of elements:

But that doesn't mean that you can't:

You can declare **vectors** in a similar manner, just don't separate items with the comma (,):

In a similar fashion you can declare **multi-dimensional arrays**:

Now we've covered some basics of basics. Up next I want to quickly cover how to handle **data** with Julia.

. . .

The Data Science Part

To begin with, let's install 3 libraries — for dealing with CSV files, data visualization, and data frames in general:

It will take a minute or so to install, but once done we'll have to import them before using:

Okay, now we're ready to go. For some quick data analysis, I will be using the **Iris dataset**, available freely on GitHub. Let's load it in and do some magic:

In case you're wondering, `normalizenames` param will ensure that column names don't have any unwanted characters, like dots (.), and those will get replaced with an underscore (_).

You can explore the data in a similar fashion as with Python's Pandas library. Let's see how to grab first 5 and last 5 rows, alongside with a quick statistical summary:

And that doesn't even scratch the surface. Anything you can think of that Pandas library has, most likely has an alternative in Julia.

And yeah, the language is also amazing plotting-wise. Take a look at what these couple of lines are able to produce:

Out of the box, it looks better than anything I've produced with **Matplotlib**. You can also do some fancy stuff, like combining plots:

And there's plenty more you could do, but that's out of the scope of this article.

. . .

Before you go

There's no doubt that Julia is an amazing language. The goal of the article was to showcase just the basics — be it of a language itself, or its capabilities to handle data.

In the following articles, I will focus on individual parts in much more depth, while also considering how it differs from Python, and whether I think the difference is positive or negative. For now, I'll say that the **data visualization** capabilities are far beyond anything traditional Python visualization libraries can offer.

How I stand on the other subjects, well, you'll have to stay tuned to find that out. Thanks for reading.