

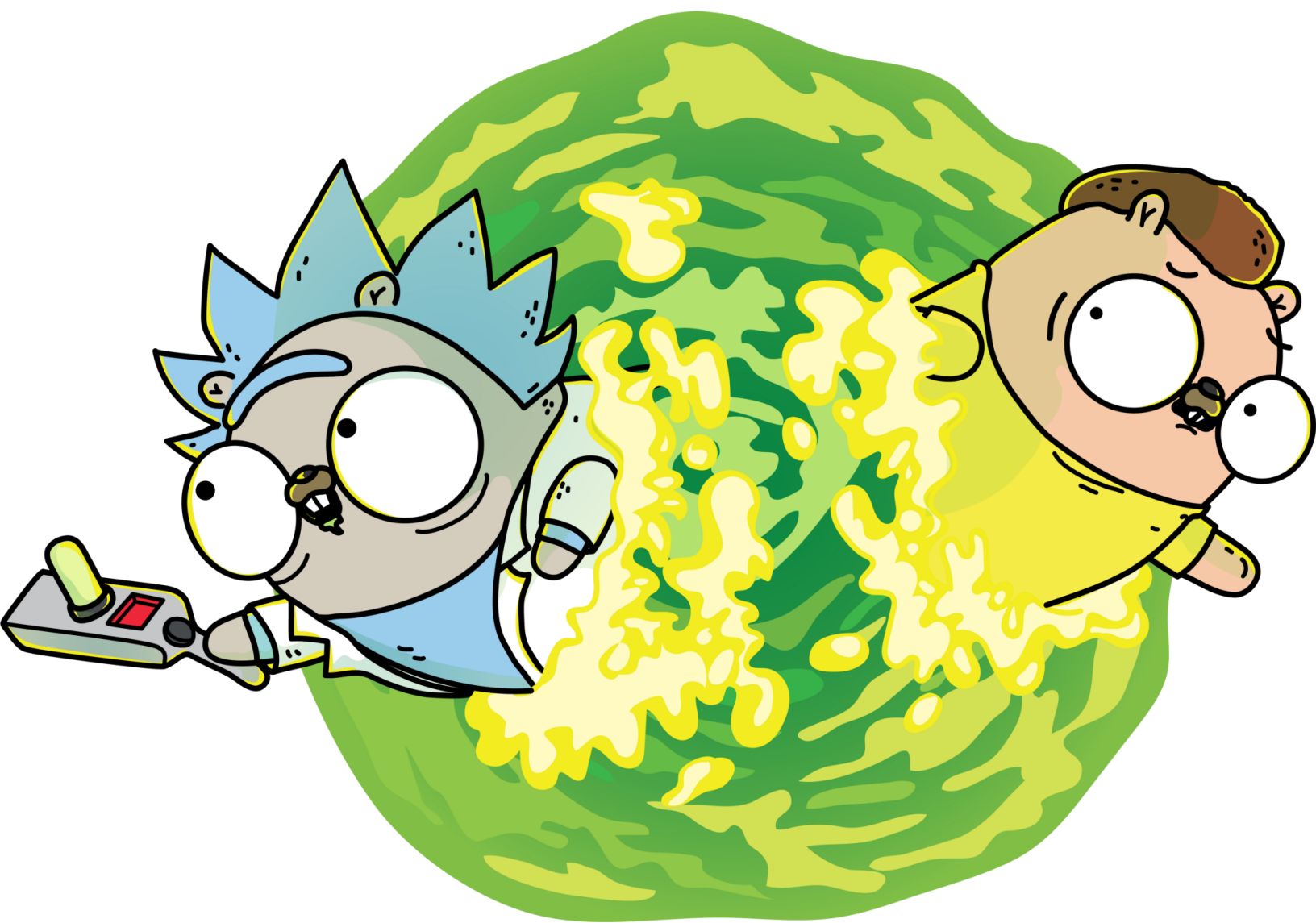




Kirill Rogovoy [Follow](#)

  Software Engineer and Traveler. Coding for fun. Javascript enthusiast. Tinkering with Golang. A lot into SOA and Docker. Architect at Velvica.
Feb 1 · 9 min read

Here are some amazing advantages of Go that you don't hear much about



Artwork from <https://github.com/ashleymcnamara/gophers>

In this article, I discuss why you should give Go a chance and where to start.

Golang is a programming language you might have heard about a lot during the last couple years. Even though it was created back in 2009, it has started to gain popularity only in recent years.



Golang popularity according to Google Trends

This article is not about the main selling points of Go that you usually see.

Instead, I would like to present to you some rather small but still significant features that you only get to know after you've decided to give Go a try.

These are amazing features that are not laid out on the surface, but they can save you weeks or months of work. They can also make software development more enjoyable.

Don't worry if Go is something new for you. This article does not require any prior experience with the language. I have included a few extra links at the bottom, in case you would like to learn a bit more.

We will go through such topics as:

- GoDoc

- Static code analysis
- Built-in testing and profiling framework
- Race condition detection
- Learning curve
- Reflection
- Opinionatedness
- Culture

Please, note that the list doesn't follow any particular order. It is also opinionated as hell.

GoDoc

Documentation in code is taken very seriously in Go. So is simplicity.

GoDoc is a static code analyzing tool that creates beautiful documentation pages straight out of your code. A remarkable thing about GoDoc is that it doesn't use any extra languages, like JavaDoc, PHPDoc, or JSDoc to annotate constructions in your code. Just English.

It uses as much information as it can get from the code to outline, structure, and format the documentation. And it has all the bells and whistles, such as cross-references, code samples, and direct links to your version control system repository.

All you can do is to add a good old `// MyFunc transforms Foo into Bar` kind of comment which would be reflected in the documentation, too. You can even add code examples which are **actually runnable** via the web interface or locally.

GoDoc is the only documentation engine for Go that is used by the whole community. This means that every library or application written in Go has the same format of documentation. In the long run, it saves you tons of time while browsing those docs.

Here, for example, is the GoDoc page for my recent pet project: [pullkee—GoDoc](#).

Static code analysis

Go heavily relies on static code analysis. Examples include godoc for documentation, gofmt for code formatting, golint for code style linting, and many others.

There are so many of them that there's even an everything-included-kind-of project called gometalinter to compose them all into a single utility.

Those tools are commonly implemented as stand-alone command line applications and integrate easily with any coding environment.

Static code analysis isn't actually something new to modern programming, but Go sort of brings it to the absolute. I can't overestimate how much time it saved me. Also, it gives you a feeling of safety, as though someone is covering your back.

It's very easy to create your own analyzers, as Go has dedicated built-in packages for parsing and working with Go sources.

You can learn more from this talk: [GothamGo Kickoff Meetup: Go Static Analysis Tools](#) by Alan Donovan.

Built-in testing and profiling framework

Have you ever tried to pick a testing framework for a Javascript project you are starting from scratch? If so, you might understand that struggle of going through such an analysis paralysis. You might have also realized that you were not using like 80% of the framework you have chosen.

The issue repeats over again once you need to do some reliable profiling.

Go comes with a built-in testing tool designed for simplicity and efficiency. It provides you the simplest API possible, and makes minimum assumptions. You can use it for different kinds of testing, profiling, and even to provide executable code examples.

It produces CI-friendly output out-of-box, and the usage is usually as easy as running `go test`. Of course, it also supports advanced features like running tests in parallel, marking them skipped, and many more.

Race condition detection

You might already know about Goroutines, which are used in Go to achieve concurrent code execution. If you don't, here's a really brief explanation.

Concurrent programming in complex applications is never easy regardless of the specific technique, partly due to the possibility of race conditions.

Simply put, race conditions happen when several concurrent operations finish in an unpredicted order. It might lead to a huge number of bugs, which are particularly hard to chase down. Ever spent a day debugging an integration test which only worked in about 80% of executions? It probably was a race condition.

All that said, concurrent programming is taken very seriously in Go and, luckily, we have quite a powerful tool to hunt those race conditions down. It is fully integrated into Go's toolchain.

You can read more about it and learn how to use it here: [Introducing the Go Race Detector—The Go Blog](#).

Learning curve

You can learn ALL Go's language features in one evening. I mean it. Of course, there are also the standard library, and the best practices in different, more specific areas. But two hours would totally be enough time to get you confidently writing a simple HTTP server, or a command-line app.

The project has marvelous documentation, and most of the advanced topics have already been covered on their blog: [The Go Programming Language Blog](#).

Go is much easier to bring to your team than Java (and the family), Javascript, Ruby, Python, or even PHP. The environment is easy to

setup, and the investment your team needs to make is much smaller before they can complete your first production code.

Reflection

Code reflection is essentially an ability to sneak under the hood and access different kinds of meta-information about your language constructs, such as variables or functions.

Given that Go is a statically typed language, it's exposed to a number of various limitations when it comes to more loosely typed abstract programming. Especially compared to languages like Javascript or Python.

Moreover, Go doesn't implement a concept called Generics which makes it even more challenging to work with multiple types in an abstract way. Nevertheless, many people think it's actually beneficial for the language because of the amount of complexity Generics bring along. And I totally agree.

According to Go's philosophy (which is a separate topic itself), you should try hard to not over-engineer your solutions. And this also applies to dynamically-typed programming. Stick to static types as much as possible, and use interfaces when you know exactly what sort of types you're dealing with. Interfaces are very powerful and ubiquitous in Go.

However, there are still cases in which you can't possibly know what sort of data you are facing. A great example is JSON. You convert all the kinds of data back and forth in your applications. Strings, buffers, all sorts of numbers, nested structs and more.

In order to pull that off, you need a tool to examine all the data in runtime that acts differently depending on its type and structure. Reflection to rescue! Go has a first-class reflect package to enable your code to be as dynamic as it would be in a language like Javascript.

An important caveat is to know what price you pay for using it—and only use it when there is no simpler way.

You can read more about it here: [The Laws of Reflection—The Go Blog](#).

You can also read some real code from the JSON package sources here: [src/encoding/json/encode.go—Source Code](#)

Opinionatedness

Is there such a word, by the way?

Coming from the Javascript world, one of the most daunting processes I faced was deciding which conventions and tools I needed to use. How should I style my code? What testing library should I use? How should I go about structure? What programming paradigms and approaches should I rely on?

Which sometimes basically got me stuck. I was doing this instead of writing the code and satisfying the users.

To begin with, I should note that I totally get where those conventions should come from. It's always you and your team. Anyway, even a group of experienced Javascript developers can easily find themselves having most of the experience with entirely different tools and paradigms to achieve kind of the same results.

This makes the analysis paralysis cloud explode over the whole team, and also makes it harder for the individuals to integrate with each other.

Well, Go is different. You have only one style guide that everyone follows. You have only one testing framework which is built into the basic toolchain. You have a lot of strong opinions on how to structure and maintain your code. How to pick names. What structuring patterns to follow. How to do concurrency better.

While this might seem too restrictive, it saves tons of time for you and your team. Being somewhat limited is actually a great thing when you are coding. It gives you a more straightforward way to go when architecting new code, and makes it easier to reason about the existing one.

As a result, most of the Go projects look pretty alike code-wise.

Culture

People say that every time you learn a new spoken language, you also soak in some part of the culture of the people who speak that language. Thus, the more languages you learn, more personal changes you might experience.

It's the same with programming languages. Regardless of how you are going to apply a new programming language in the future, it always gives you a new perspective on programming in general, or on some specific techniques.

Be it functional programming, pattern matching, or prototypal inheritance. Once you've learned it, you carry these approaches with you which broadens the problem-solving toolset that you have as a software developer. It also changes the way you see high-quality programming in general.

And Go is a terrific investment here. The main pillar of Go's culture is keeping simple, down-to-earth code without creating many redundant abstractions and putting the maintainability at the top. It's also a part of the culture to spend the most time actually working on the codebase, instead of tinkering with the tools and the environment. Or choosing between different variations of those.

Go is also all about "there should be only one way of doing a thing."

A little side note. It's also partially true that Go usually gets in your way when you need to build relatively complex abstractions. Well, I'd say that's the tradeoff for its simplicity.

If you really need to write a lot of abstract code with complex relationships, you'd be better off using languages like Java or Python. However, even when it's not obvious, it's very rarely the case.

Always use the best tool for the job!

Conclusion

You might have heard of Go before. Or maybe it's something that has been staying out of your radar for a while. Either way, chances are,

Go can be a very decent choice for you or your team when starting a new project or improving the existing one.

This is not a complete list of all the amazing things about Go. **Just the undervalued ones.**

Please, give Go a try with A Tour of Go which is an incredible place to start.

If you wish to learn more about Go's benefits, you can check out these links:

- Why should you learn Go?—Kaval Patel—Medium
- Farewell Node.js—TJ Holowaychuk—Medium

Share your observations down in the comments!

Even if you are not specifically looking for a new language to use, it's worth it to spend an hour or two getting the feel of it. And maybe it can become quite useful for you in the future.

Always be looking for the best tools for your craft!

. . .

If you like this article, please consider following me for more, and clicking on those funny green little hands right below this text for sharing. 🙌🙌🙌

Check out my Github and follow me on Twitter!

