**Go’s design philosophy**

A couple of things really shined to me from Rob Pike’s [Simplicity is Complicated](https://talks.golang.org/2015/simplicity-is-complicated.slide#18) talk.

a) **Traditionally, every other language wants to just keep adding new features**. This way, all of the languages are just adding bloatedness, too much complexity in their compilers and their specification. If this continues, every language will look like the same in the future because every language will keep adding features it doesn’t have. Consider, JavaScript adding object oriented features. The Go authors deliberately did not include a lot of features in the language. **Only those features were included for which there was consensus from the authors, for the ones which really felt like they did brought value into what can be achieved by the language.**

b) **Features are like orthogonal vectors in a solution space.** What’s important is the ability to pick and combine different vectors for your use case. **And those vectors should just work with each other naturally. Means that every feature of the language should work predictably with any other.** This way, those set of features cover the whole solution space. Implementing all these features, which work very naturally with each other, brings a lot of complexity into the language implementation. But the language abstracts the complexity and provides you with a simple, easy to understand interface. Therefore, simplicity is just the art of hiding complexity :)

c) **The importance of readability(simplicity) is often underrated.** Readability is critical, arguably, one of the most important things in designing programming languages, since the importance and also the cost of maintaining software is high. Too many features hurt the readability of a language.

* If a language has too many features, you waste time choosing which ones to use.
* The code is harder to understand simply because it is using a more complex language.

A few simple things in Go

garbage collection

goroutines

constants

interfaces

packages

Each hides complexity behind a simple facade.

d) **Reliability.** If a language is complicated, you must understand more things to read and work on the code. Similarly, to debug it and to be able to fix it. This also means that new developers on your team will need much larger scale up times, to get their understanding on the language upto the point at which they can contribute to your codebase.

* Readable code is reliable code.
* It's easier to understand.
* It's easier to work on.
* If it breaks, it's easier to fix.

If the language is complicated:

* You must understand more things to read and work on the code.
* You must understand more things to debug and fix it.

The four important things a language must thrive to achieve is

* speed/efficiency
* reliability
* scale
* simplicity.

If we consider languages like C or C++, they are great at *speed*, *scale*and *reliability* but in terms of *simplicity*, they are not so great.

 Java on other hand is very *reliable* and highly *scalable*, but moderately *simple* to write and not so *efficient* compared to other low-level languages.

Python is an widely adopted language and very *simple* to write but not so *efficient* and *reliable*.

Whenever Google creates a language or a framework, I suppose it’s because they want to eventually use it in their own organization. Angular, Material design, Dart, Flutter etc. are few examples.

Things that make Go a great language is its simple concurrency model, its package based code management and its non-strict (*type inference*) typing system. Go does not support out of the box Object Oriented Programming experience, but is support structures (*structs*) which with the help of methods and pointers can help us achieve the same.

We also thought that it was important to provide language-level support for concurrency. Multiprocessors are easier to program when the language knows about stack and memory management, and when it provides messaging and synchronization primitives.