Loc Nguyen

Professor Raheja

CS.4080.03

Evaluation of Go Programming Language

Go or Golang is a newly established language started development in September 2007 with three original developers: Swiss computer scientist Robert Griesemer, who worked on Google V8 JavaScript engine, Sawzall language, and Java HotSpot virtual machine; Canadian programmer Rob Pike, who worked at Bell Labs on the Unix team and was involved with the creation of Plan 9, Inferno OS, Limbo programming language, and co-creator for UTF-8; and lastly, American computer scientist Ken Thompson, who also worked at Bells Labs and was involved in B and C programming language, Plan 9, and was dubbed “Father of Unix” alongside Dennis Ritchie for his work on designing and implementing the original Unix OS.

The creators of Go wanted something that was expressive like Python or JavaScript but lightweight and efficient like C or C++. There were few IDEs available at the time and production software was mostly written in C++ and Java. The triangle of tribulation of choosing only two out of three conditions of efficient compilation, efficient execution, or ease of programming, meant that developers had to sacrifice and compromise. For example, they would choose ease of programming over safety or efficiency by using dynamically typed language like Python. The moment was ripe for Go to address these flaws by combining the ease of dynamically type language with safety and efficiency of statically typed, compiled language. Additionally, Go features such as concurrency, automatic garbage collection, and rigid dependency specification provide better design system than any existing libraries or tools at the time.

To maintain efficiency, Go is excellent at compilation speed, boasting a compilation speed that’s even faster than C and a lot of compiled language. This allows programmers and developers to quickly iterate, test, and deploy code to production which decreases downtime versus something like Java which takes a while to compile huge programs. To achieve this, creators of Go decided to make Go an imperative and procedural programming language, mixing in a little bit of functional programming language features such as first class and lambda functions. That means there are no classes, objects, or features like inheritance from object-oriented programming languages. However, developers can mimic some functionalities with interface and struct in Go.

First, like C, Go must import packages like “fmt” which handles input/output stream operation or other relevant packages such as “math” for specialized mathematical operations. The main function must be defined but with minimal boilerplate code, like Python, which increases writability. Syntax and scopes in Go is eerily similar to what we

see in Python and C/C++ where it supports single-line multivariable declaration or initialization of the same type, out of scope once function finishes, and type inferencing where, instead of the keyword “var” follow by “=” to specify the variable type, Go uses “:=” for assignment operation and guess the type based on assigned value. It’s important to note that variables are still statically typed and checked at compiled time, unlike Python.

Data types in Go include basic (numbers, strings, Boolean), aggregate (array and structure), reference (slice, map, function, pointers, and channel), and interface. The three basic data types are typical and almost universal among programming languages. Aggregate type struct store collection of heterogeneous fields into a single field and array is mutable, fixed size, and less popular than slices. Function in Go is very useful and intuitive, allowing for “naked” return statement where return variables as specified before function is defined. Same for pointers, which may be used to access field values in a struct, mimicking OOP properties. Slices are dynamically sized and are more popular than arrays due to its flexibility so operations on slices are done to speed up execution and decrease memory usage. Map behaves like a dictionary in Python where it maps a key to a value, but a notable keyword “make” must be explicitly called to initialize a map “object” or a channel. The make keyword is implicit called for pointers, functions, slices, arrays, and various other structures but in this case, it must be called explicitly. This decreases uniformity and regularity in Go because of special cases such as this.

The most special feature of Go is channel, a typed conduit which you can send and receive values using the channel operator <- and the “make” keyword with “chan”. Channels are often used in goroutines, a way to speed up program execution speed. Instead of using return statement, data is passed via the channel to avoid costly execution time of variable assignment of return value. This is the concept of concurrency, asynchronous multitask execution in Go. Concurrency in Go is very useful for API calls, data aggregation, and other tasks that is repetitive yet require speed. As a side note, concurrency and parallel computing or parallelism isn’t the same. Concurrency in Go will execute the same task simultaneously on many threads in comparison to parallel processing where many threads work together to execute one task simultaneously. This model of concurrency works best when there are many tasks with small execution time and will suffer when it crosses into territory of parallel processing. If concurrency is used to execute many huge tasks simultaneously on many threads, it would cause a meltdown in the program.

In our demo, we decided to show the difference between execution time with and without goroutines by calling the OpenWeather API to return the temperature data for 5 cities throughout the world. The average execution time without goroutines were around 400 to 500ms whereas the average time with goroutines sits in the high 100 to low 200ms range. So, even with just 5 items in the program calling the API, goroutines are already about twice and approaching triple the execution speed when a program uses a

regular function to return value. However, since tasks are executed asynchronously and independently of each other, we had to use a library in Go called “Wait Group” to tell the loop of goroutines to wait for all threads to finish execution before returning data through the channel. Otherwise, the output data format will not match predefined print statements and cause jumbling on the output stream.

Interface in Go is quite special as it is implemented implicitly by default. There is no “implement” keyword to specify which method or function definition implement which interface predefined method signature. Under the hood, it’s a tuple of value and concrete type: (value, type) that make up the method signature. Go will automatically determine implementation based on signature and able to make inferences about interface function definitions scopes. This is a hit to readability but increases writability. If developers want to “know” which functions implement which interface, they have no choice but to match the function signatures and figure it out themselves. This is not much of a hassle when we were developing our demo code since there was only one interface with 2 methods. However, as the code base increases or program lengthen, it is a huge challenge to troubleshooting and maintenance when programs break, and developers have to comb through thousands of lines and individually matches function signatures to interface in order to find bugs and code trace. However, it is great for writability because if developers already know the signatures in their head, it’s a breeze to write the methods with fewer boilerplate code.

Control structures in Go is very basic and primitive with normal if-else, switch statement with automatic break so there’s no “break” keyword but the “fallthrough” keyword exist to negate the automatic break, and normal iterative for loop. At first, it was weird, but I’ve grown to like the generality of Go where it will use for loop as while loop. There is no “while” keyword in Go so a modified for loop is used as a substitute that acts the same way a while loop would. If we omit the start and step and only check the conditions, that’s pretty much a while loop in Go. This special feature increases both readability and writability if you’re used to the syntax because it reduces the complexity of vocabulary or unexpected behavior between using different control structures. Another special feature of the for loop is the use of blank identifiers, similar to Python, where an underscore _ is used to denote unwanted iterative variables. For example, if you’re looping through an array, you may want to use the index but not the element or vice versa, you can use the blank identifier to signal to other developers that this iterative array variable is not important, increasing readability.

Another special feature of Go is the ”defer” keyword which delay the execution of a function until the surround function returns. Often, this is use in conjunction with goroutines and Wait Group to delay output stream and contribute a lot to formatting. The implementation of defer is Last In First Out (LIFO) like a stack ADT.

Exception handling is quite crude but often good enough to handle most errors. There is no “try-catch” or anything complicated. Instead, Go developers uses if statement to check for error and which type of error. They do this by using single line multivariable

assignment of the variable that will hold the data and an “error” variable that’s implicitly assigned the keyword “nil” since it wasn’t explicitly assigned any value. Side note, nil is another special feature in Go where it doesn’t have a default type and is untyped, unlike Null in other languages where Null is its own type. Nil represents zero value in pointers, variables, map, slice, functions, and so on. Back on track, developers would check whether or not the error variable is equal to nil, meaning there’ s no exception, or if it’s not equal to nil, meaning there is an exception, and we can output the variable to find out what’s the problem.

Taking a bird-eye view, the readability of Go pretty alright with good simplicity, orthogonality between different features through type inferencing or sharing implementation of different control structures, and some level of uniformity throughout the syntax and flow of program. It avoids unnecessary symbols or complex constructs to reduce cognitive load on developers and syntax or scopes is defined via curly braces with unheard/optional choice to use semicolon. At compile time, Go will automatically insert curly braces. Go holds a tremendous amount of similarity versus Python, C, and C++ which boost cross-language understanding and make Go easy to pickup and learn. However, Go introduces new concepts of defer, wait group, and channel that may confuse new developers that first start learning about Go.

Hand in hand with readability is writability, which is great as it is very expressive and get the hang of eventually. With type inference, developers can quickly write programs in combination with familiarity from other languages and with minimal boilerplate code, unlike Java and much more like Python. Packages in Go is one-word short and simple. Also, Go encourages developers to explicitly do error handling with conditional if so error will always return explicitly and problems are easily identified. This approach improve the writability and reliability of code as they are more robust.

Lastly, Go is a reliable language for backend or microservices that need speed and simplicity. We have explore the advantages of Go speed comes from concurrency and goroutines while being easy to learn. The execution speed of Go is slower than natively C or C++ because of the small overhead of Go runtime, a small snippet of code that manages automatic garbage collection and memory at runtime. Another nice feature in Go is built-in testing with “go test” where Go will do unit testing without using add-on packages like Java. However, there’s no free lunch and Go do come with some drawbacks. Being a newly developed language for a little over 10 years ago, Go lack a robust library of packages that is robust to further speed up development time unlike Python where there’s plenty of useful packages such numpy, pandas, re, and many others that work together to make Python code robust and abstracted away complexity. A debatable disadvantage of Go is the type safety of static typing where Go trade away some flexibility. Finally, the Go developers community is growing but remain small since Go has no other applications beside backend services which is an important but small niche. It would be hard to find collaborators or companies to work with using Go beside Google.

Overall, my experience of learning and programming in Go brought me an incredible amount of experience in applying software engineering principles to showcase the applications of Go. I enjoyed the slow, painful, but ultimately, rewarding learning experience that the project has given me.

Works Cited

Bai, Yaroslav. "Why Use Golang For Your Project." 27 June 2023, www.uptech.team/blog/why-use-golang-for-your-project. Accessed 12 Dec. 2023.

Barney, Nick. "Go programming language." Edited by Alexander S. Gillis. *IT Operations*, TechTarget, www.techtarget.com/searchitoperations/definition/Go-programming-language. Accessed 12 Dec. 2023.

"Go by Example: Structs." *Gobyexample*, gobyexample.com/structs. Accessed 12 Dec. 2023.

"Go for Loop." *Programiz*, www.programiz.com/golang/for-loop#:~:text=Golang%20Blank%20Identifier%20in%20for,identifier%20_%20to%20avoid%20this%20error. Accessed 12 Dec. 2023.

Jha, Indra Nand. "Exploring the Pros and Cons of Go (Golang) as a Programming Language." *LinkedIn*, 23 Mar. 2023, www.linkedin.com/pulse/exploring-pros-cons-go-golang-programming-language-indra-nand-jha/. Accessed 12 Dec. 2023.

"Ken Thompson." *Wikipedia*, Wikipedia Foundation, 12 Aug. 2021, en.wikipedia.org/wiki/Ken_Thompson. Accessed 12 Dec. 2023.

"Robert Griesemer." *Wikipedia*, Wikipedia Foundation, 20 Aug. 2020, en.wikipedia.org/wiki/Robert_Griesemer. Accessed 12 Dec. 2023.

"Rob Pike." *Wikipedia*, Wikipedia Foundation, 9 Aug. 2019, en.wikipedia.org/wiki/Rob_Pike. Accessed 12 Dec. 2023.

Saini, Ankita. "Arrays in Go." *GeeksforGeeks*, GeeksforGeeks.Org, 6 Sept. 2022, www.geeksforgeeks.org/arrays-in-go/. Accessed 12 Dec. 2023.

---. "Interfaces in Golang." *GeeksforGeeks*, GeeksforGeeks.Org, 20 Nov. 2019, www.geeksforgeeks.org/interfaces-in-golang/. Accessed 12 Dec. 2023.

"Welcome to a tour of Go." *A Tour of Go*, Go.dev, go.dev/tour/list. Accessed 12 Dec. 2023.

"Why You Should Go with Go for Your Next Software Project." *ScaleFocus*, 29 Aug. 2022, www.scalefocus.com/blog/why-you-should-go-with-go-for-your-next-software-project#:~:text=As%20with%20any%20programming%20language,take%20a%20lot%20of%20effort. Accessed 12 Dec. 2023.