Go

John Hossler

December 1, 2017

Hello, my name is John Hossler. I realize that sometimes the choice of language that is used in a project is restricted to what what has been built before, but my hope is from this presentation is to perk enough interest in developers to play around with it and see it as a candidate for a new project where the language isn't nailed down.

Of course, in the span of five minutes I can't cover everything that is great about Go, so I'll focus on some things I found neat about Go. Go has a style builtin to the language, it takes advantage of interfaces for object oriented programming, and it is easy to write but still a compiled language.

As far as builtin style, I'll first talk about some of the built-in tools Go gives you to format your code to a standard. Gofmt helps cleanup code to make it more readable for humans. With this tool, I end up spending less time formatting things like long dictionaries or comments because Go does it for me, as well as lining up declarations to increase readability. One thing that I'll show a bit more of in a second is that Go uses what other languages would consider style as syntactic sugar. For example, Functions that are capitalized in a package are your public functions, and ones that aren't capitalized are local only to that file.

When compared to other languages, I have found Go is quite strict about what is allowed to compile. We all know that there are different ways to write if statments in other languages, depending on the placement of curly braces, but in Go, this won't compile. It raises an error, and if I fix this issue and try to recompile, Go foils me again! I had an unused variable, so let me go ahead and fix this – and now it compiles and I can run it. Here, I haven't set up my editor, but the gofmt tool will even use the Go preferred tab length of 8 and clean up my code. This is just a minimal example, but when code is being worked on by a lot of people, having these tools to clean up code help keep code reviews small and focused on a feature rather than formatting issues and unused code.

Go also took a different approach for enabling code re-use by using interfaces. As an example, let's talk use a circle and a rectangle. I define a geometry interface, and say that anything with an area and a perimeter function that returns a float64 will satisfy this interface. After defining the rectangle and circle, which look completely different, I can create a function and give the parameter a interface type. This means anything that satisfies it can be passed into this function. I can run this, and I get the expected output. The reason I think this is so powerful is that the interface is implicitly satisfied, so if I made a package that imported the geometry interface, but created a square object instead, I could still pass my square object into the measure function as long as my square object has those functions. With interfaces, a developer defines what interface a function needs, and anyone can create an object to satisfy that and reuse that function.

While this is a compiled language, it has type inference built in so that it is easy to write (and read) like an interpreted language. Another item I would love to have touched on more is defer statements in go. They allow you to specify actions to take that are to be done at the end of the function, regardless of if an error is thrown before. This makes it easy to ensure files are closed properly at the end of reads, or any other connection that needs to be properly destroyed is destroyed.

I wish I had signed up for a longer time slot, because there are so many more great things I love about Go. I encourage you to use it for a programming challenge, like adventofcode.com, and give learning it a chance. Thanks!