EJ Cervantes

Dr. Liz Johnson

CSCI 300: Programming Languages

19 November 2019

Case Study: Go

Go also referred to as Golang began development in 2007 after being developed by Robert Griesemer,Rob Pike, andKen Thompson. As the name may imply, it was developed under the authority of Google to suit their engineers’ needs and also address the criticisms of other different languages they used. (Golang: How It Started - A History of Success)The three designers and project leader for the creation of Go were inspired by their gripes with C++. It wasn’t until 2009 that Go was publicly announced and then another 3 years until it was publicly released. Since the initial release on March 28, 2012, Go has had 13 major releases and current on release 1.13.1. (Documentation)

The main reason for the creation of Golang was to increase programming productivity especially within the developers at Google. The mission of Golang was to combine the ease of using all multi-core usage, integrating networked machines, all while not having confused users dealing with large collections of source code. This included taking the elements of readability and writability from Python, and combining it with the static typing and run-time speed of C++. (The Evolution of Go) Because of this, Go is shown to be a great choice for creating large scale, complex software. Because of this Go, is considered a multi-paradigm language touching upon concurrent, functional, imperative, and, of course object-oriented.

Within Go there are two different types of variables, global and local. Go will use a stack for local and a heap for the global variables. Some variables within Go are determined at compile time making them static variables. (Documentation) Scope being determined by {} determines whether the scope for a variable is global or local. All variables outside any functions or blocks are labeled to be global without any declaration. Variables listed inside blocks and functions are automatically labeled as local.(Scope of Variables in Go)

Local variables within a function can still be accessed inside another nested block within that same function. Variables with the same name inside the same scope are not allowed and will produce a compile-time error since variables are static. Global variables are accessible throughout the lifetime of the program. Local variables are still allowed to have the same name as a variable with a global scope, but preference is given to the local variable if it is in local scope by the compiler.(Documentation)

Binding is defined as an association between an entity and an attribute. For example, an entity can be something like an object while its attributes can include name, value, type, and scope. Storage binding refers to how these attributes are allocated and deallocated in memory for a certain object. Within Go, there are two different storage types available to use, a local stack and a global heap. Go prefers to utilize allocation on the stack because it is inexpensive and only requires two CPU operations. To belong to the local stack, it is required that the lifetime of the variable in question be determined during compile time. All other allocations are to the heap at run-time. Go does automatic memory allocation and deallocation/garbage collection for users, in other words memory management is taken care of in as much of an efficient way possible for the compiler to handle our created memory blocks. Thus the user does not need to worry about allocations to the heap, other than the fact it is significantly more expensive than utilizing the stack. (Allocation efficiency in high-performance Go services)

The Go programming language is not overflowing with types and selection of what types to choose. Instead, Go has four major classifications of types that can be explored according to *www.tutorialspoint.com*. These types include Boolean, Integers, String, and Derived Types. Most of these are very self explanatory on what data types are included under these classifications. Something that was unlike most programming languages, Go makes an effort on the developer end to think about what type of integer to use, signed versus unsigned, and exactly how many bits to make the integer. For example **uint32** is an unsigned 32-bit integer (0 to 4294967295). This is very interesting as it could cause overflow if the programmer is not careful on what values they are using, but would also increase the efficiency and correctness of the program if done in a correct manner.

Another interesting area to explore is that of their derived data types. They have many derived types such as pointers, arrays, union, map, and channel. Two of the most interesting aspects over these derived types are a slice, and channel. A slice is a different abstraction of an array and has different functionality added to an array that is helpful in most cases. Because of this slices are almost always preferred over arrays in Go. Arrays in Go let you hold a predetermined number of data types together but cannot increase in size dynamically if you would like to increase the array, and it does not let you pull sub-arrays of your own. Slices are able to be declared without a specified size as you are able to append() types to the slice. Slices are also always defined as NIL if they have not originally been given a certain size. Channels are a typed conduit in which you are able to send out values, and receive values. Channels are used for synchronization between separate threads managed by Go, they are referred to as Goroutines. Goroutines are light-weight threads and cost significantly less than a thread so it is very common to have hundreds of goroutines running concurrently.(Documentation)

Go does parameter passing in two different styles. Pass by value and also pass by reference. With pass by value, the actual value of the passed parameter is copied in a separate memory location for the function to use. Therefore edits to the memory location with the function’s parameters, does not change the value of the memory location used by the caller. In other words, the “called” is changed while the caller remains. In pass by reference, we need to use Go’s derefrence operator (\*) to change the value of the one memory address.(Documentation)

Functions and control-flow within Go are very related. Functions are declared by the declaration statement of “func” followed by the name and formal parameters needed. If no parameters are needed it can be left blank. Go functions also do not require a return type as it is optional, but for safe coding one can declare the return type after the parameters. Go will begin execution at func main() and continue from there, having a return type for main is discouraged. Go uses many familiar control-flow statements found in other languages such as if, if/else, for, switch, and recursion.

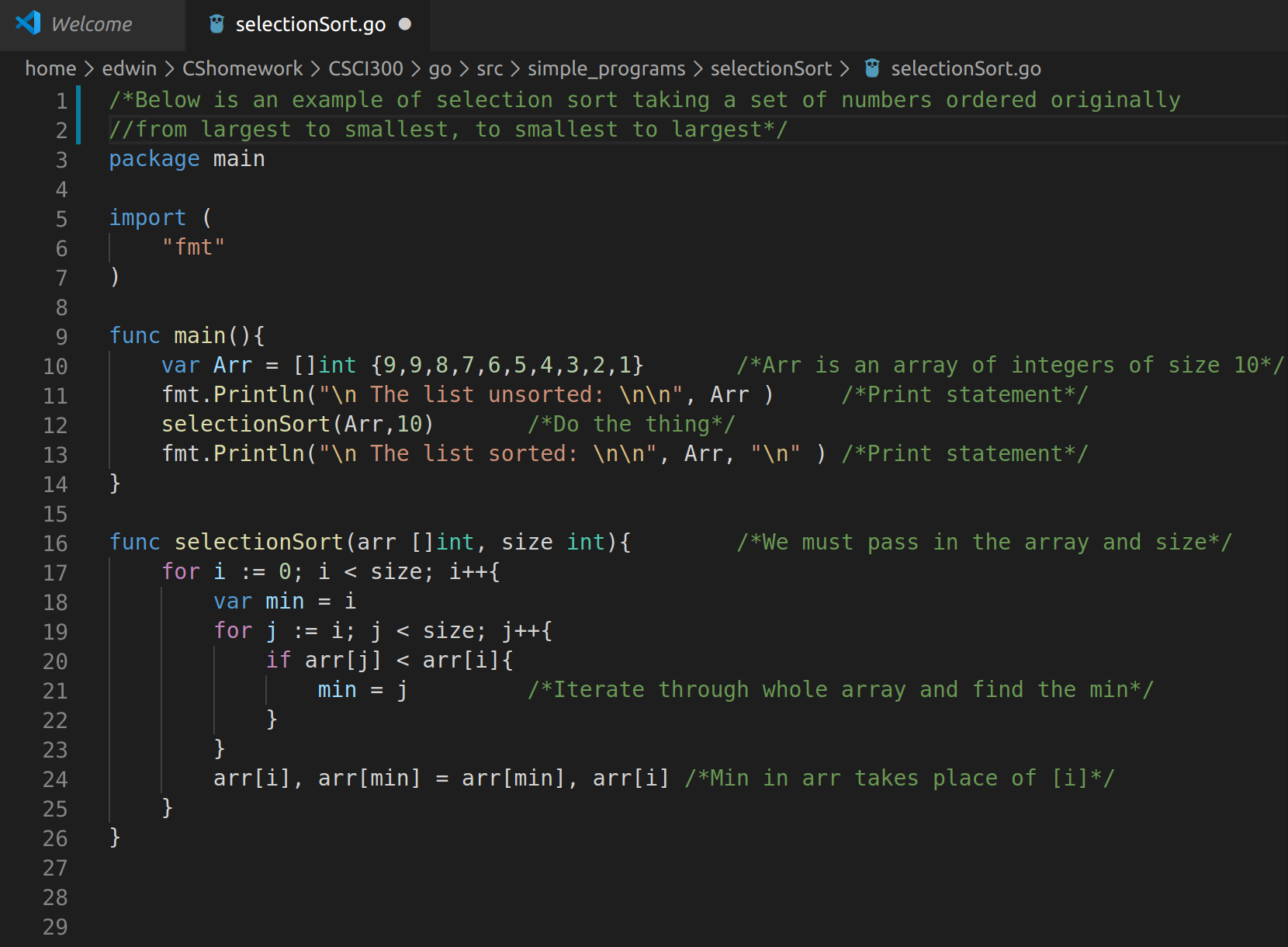
The Golang type system is very interesting case to study on its own, this is because Go puts priority on flow of the program rather than being minute with consideration of types. As side by creator Rob Pike, “… the more important idea is the separation of concept: data and behavior are two distinct concepts in Go, not conflated into a single notion of ''class ``. (IOS and Android Programming with Go). To users, one may ask what exactly this means and how to interpret it. Rob Pike is explaining how the control flow works separate from the model the user is trying to implement. To make the representation of data less of a roadblock or worry to the user, Go implements structs. A collection of fields that are used to group and pair data together to form records. This causes a lack of type hierarchy which breaks apart inheritance properties between objects which makes them resemble a data structure rather than a real world implementation.

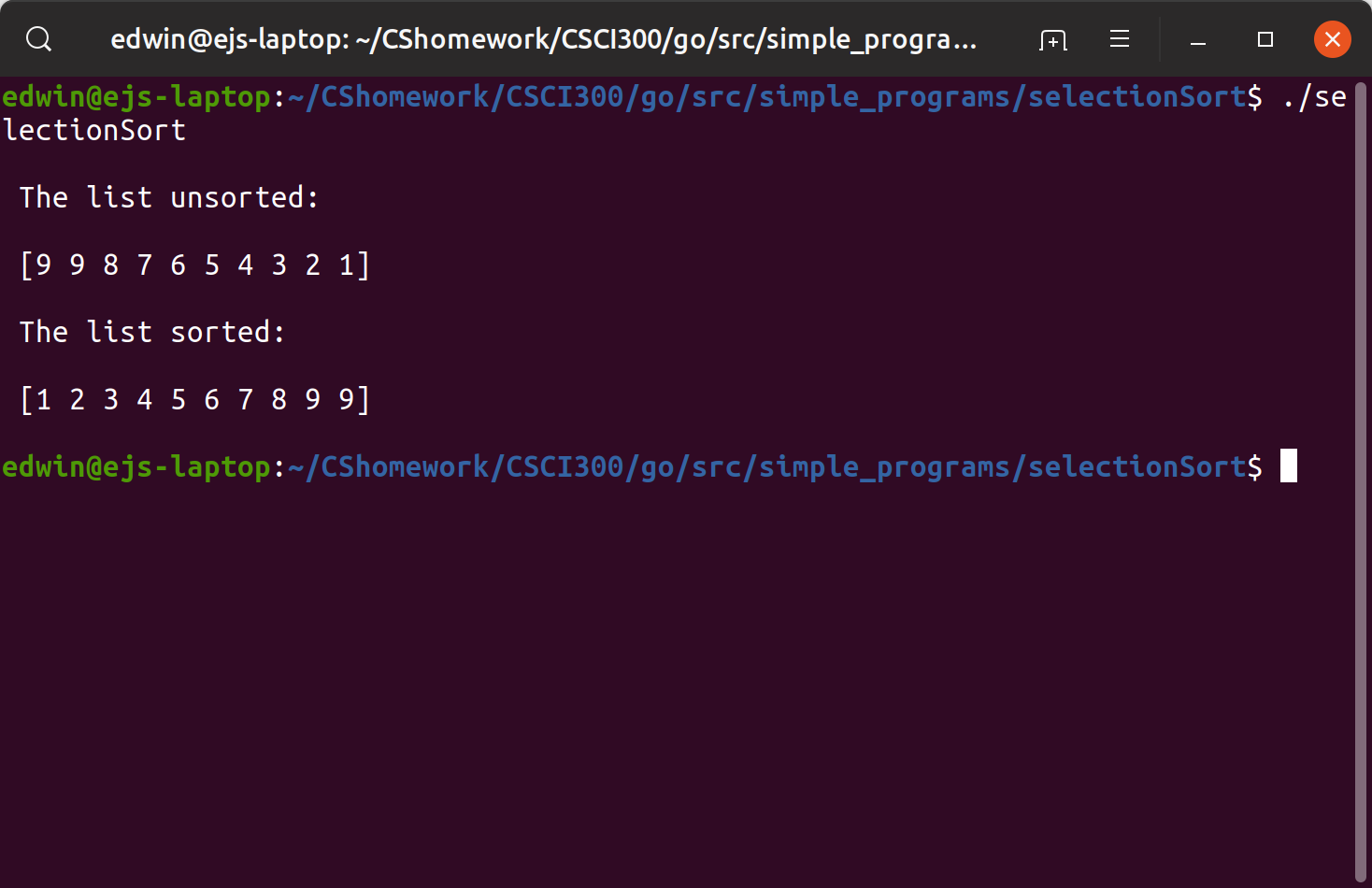
Go is a strongly typed language according to the Go documentation . This is because it protects against you messing around directly with pointers that can cause programmer error. Go is also statically typed in this case according to the documentation provided by them. This can also be shown because all variables need to be explicitly declared, thus they are locked in at compile time.(Documentation)

There are two major aspects or features of Go that make it a very useful language across many different platforms. The first example is how fast and efficient the Go compiler is. And the second is the simplicity of the language itself when it comes to either reading or writing it. The reason for the language being simple, is because of how reduced the syntax is. This reduced syntax makes it easier to compile large Go programs. Along with this, Go compiles within the native device or platform you are working on, therefore adding an environment is not needed.

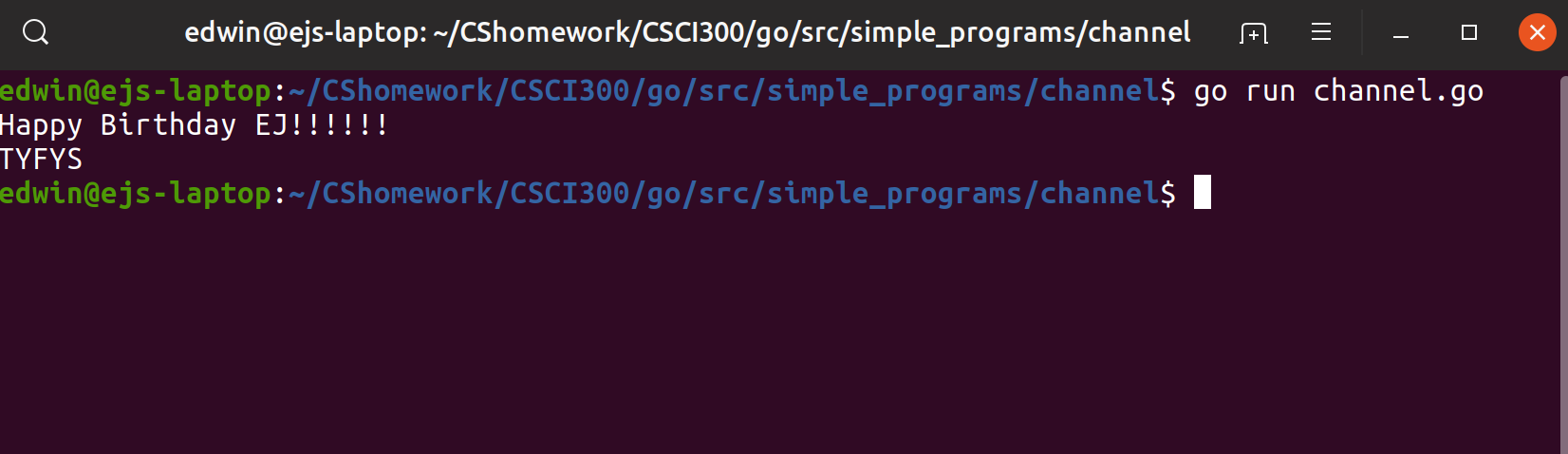
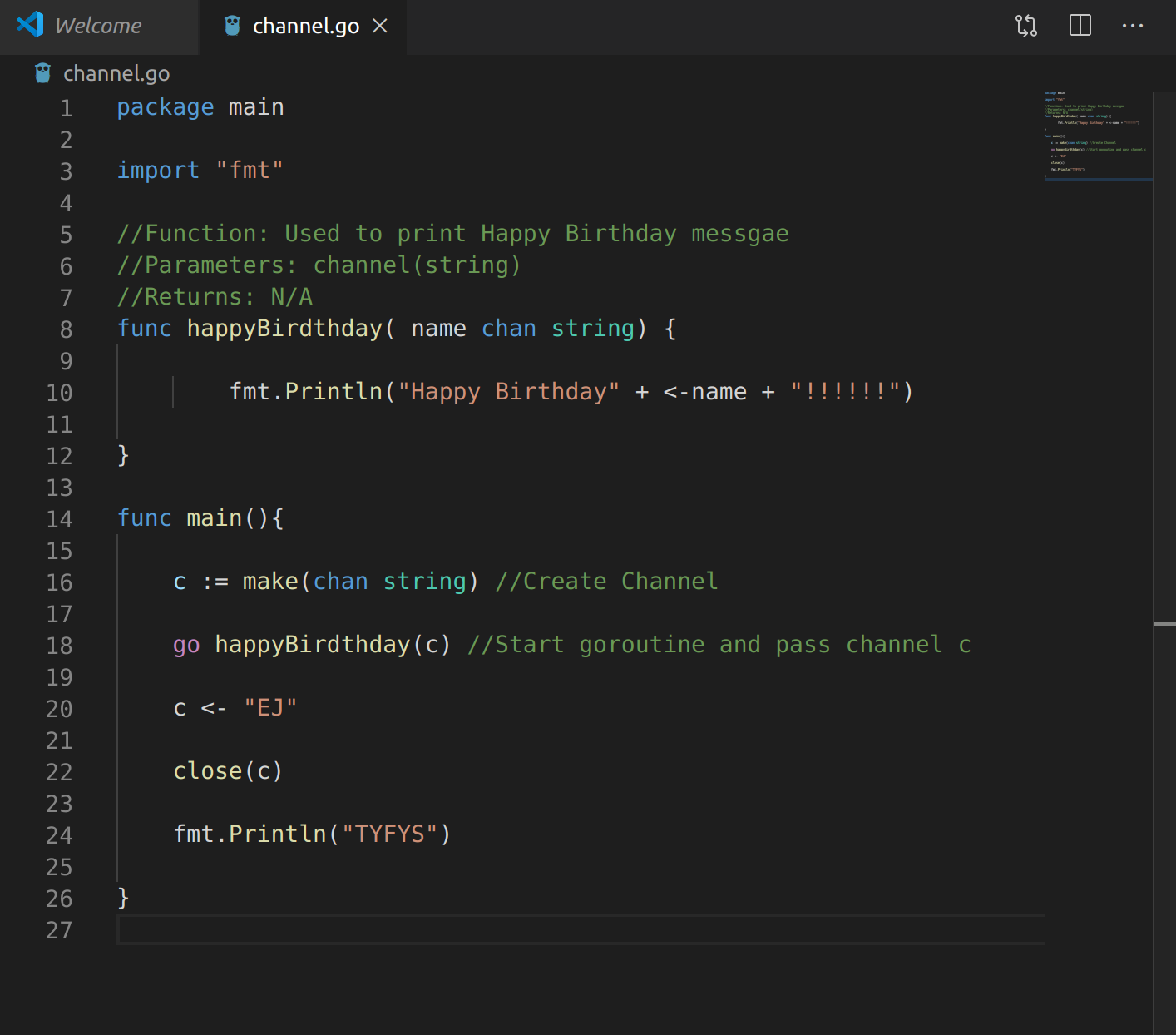
Go is very efficient at creating simple scripts with ease and very little effort. This is one of the main advantages and features of Go and why it is preferred by many for large data structures and projects. Go simplifies the trivial operations that could be a source for longer more complicated functions which are welcoming to logic errors and bugs. Below we evaluate two different examples of a Go script to show how easy and reduced the code is. All examples are my own work and written and tested in less than 10 minutes each.

**Selection Sort:**





Selection sort is a very slow sorting algorithm and especially since it behaves at the same O(n^2) no matter the case. In this example I would like to show the reduced declaration and population of our array. Another important factor that is going on here is that we are able to pass full arrays with ease (see parameter passing). The most interesting factor above all is how the equivalent of a swap() function is implemented. The swap is being performed without the utilization of a helper function or third variable holder. This is significantly less of a headache and complication than most other languages where if a swap() function is not provided, this can prove to be a struggle logically. In Go, we store the two variables into CPU registers and then write them back out to the reversed variable holders. (Memory Blocks - Go 101)

****

**Channels Explained:**

Channels are strictly used in Goroutines as a vessel of transportation to either read from, or pass data. This allows a goroutine to send data to a channel while a completely separate goroutine is still reading from that channel. Synchronization between these is what ensures the correct reading and writing of the valid data. To read or write from a channel we use ‘←’. If we wish to assign the current channel to a fixed variable, the channel would be on the right hand side of the assignment, while if we wanted to write to a channel, the channel variable is on the left hand side of the assignment. From the example above, we must create the channel first. We then initiate our goroutine by the keyword ‘go’. This means we have 2 different goroutines active. Control is sent back to main() and executes the next line. We then assign a string value of “EJ” into the channel c. Now that a channel can be passed along to our hapyBirthday() function, our scheduler takes over and gives control off to the second goroutine. After it leaves scope, we return to the goroutine we were in originally, main(). We then close the channel so no more data can be passed through. (Anatomy of Channels in Go - Concurrency in Go)

One of the most accurate ways to judge a programming language on an even basis, being compared to others is to examine the language’s readability. writability, and reliability. After having the coding examples from above, we will begin to dive into Go’s three characteristics and how each ranks. We will be using examples that we may not have covered in this paper yet but they are all valid. To quantify a rating, each category will be assigned a numerical value from 1-10 with 10 being the max score that can be attained.

**Readability -** More often than not in a work environment you are not programming. You are reading other people’s programs. This can be in the case of peer review/editing, and also seeing examples of how to implement a function. This is where readability comes into play. Readability is the level of how easy a program is able to be interpreted by a human. The further away from assembly language and closer to a natural language, the readability increases. There is a slight learning curve with the readability of Go even for an experienced programmer. This learning curve is not something that people should be wary of but rather embrace the change. The syntax and flow is very similar as compared to Python being easy on the eyes to understand and grasp what is occurring. Where the challenge comes in reading Go scripts is in learning the new terms one does not see very often and how their data types are specified. For example, the usage of channels and slices are not found in other programming languages, but there are structures similar such as a List in Python or threads in most major languages. 7/10

**Writability -** As programmers and developers, eventually we will be constructing and forming our own lines of code. Writability is the ease at which programs can be created for a specific purpose and job. Many programs can solve the same problem but there are obvious choices at which languages are easier to implement. Writing Go scripts can present a bit of a challenge to begin with. This challenge is presented by having to learn primarily through the Go documentation on how to write things because there is not much support outside of there. Go is still a very new language and rising in popularity so it is not something that has priority on tutorials, instructions, and examples of large scale projects. In terms of writability itself, the language is very compressed, usually mocking psuedo-code. As shown in the examples above, most operations can be condensed. 7.5/10

**Reliability -** Reliability sometimes can be the most important of all three depending on the circumstance. Reliability is defined by how often the program will reach the desired outcome and not stop working under special circumstances. Go has been optimized to perform better than C++ and Python taking only the wanted characteristics from both. Go is one of the main languages used at Google on a regular basis. Not only can Go handle small projects, but it is also able to be used on large full scale projects and performs better than most. It goes without saying that Go is a very reliable and trusted language as it has been vouched for by one of the largest tech companies in the world. 9/10

Using these three characteristics and final scores, we will then go ahead and evaluate how Go would function as a language of choice for three completely different situations.

1. **First Language in introductory Computer Science course**
   1. I believe that Go would make a very good option to explore while teaching an introductory course. The most important of the three characteristics to consider here is the readability of the program. The main reason for this is because how introductory level programs can be reduced in the amount of lines of code written. This makes the scripts look less intimidating to newer students because the writability and readability are increased here. Introductory students also spend more time looking at examples and reading examples of code rather than writing.
2. **Application that manages the student records of about 5000 students a year**
   1. The most important characteristic between the three here would be the reliability of the program. With so many sensitive records being recorded, the integrity of these cannot be messed with. Therefore we would need to select the most reliable way of preserving the information so that we are sure it is as up to date as it needs to be. Go contains an SQL package available for import. Meaning database support is present, without utilizing a database this problem would be almost impossible to resolve. Modeling these students would also be very easy to implement using Go’s struct type. Therefore Go would be a perfect option to attack this project.
3. **Development of a mobile app to report the closest bus stop to the user's current location**
   1. Because Go is so scalable and easy to follow as a program extends, Go is also a very good option for a mobile app. Creating a data structure to locate the closest bus stop in Go would be easy to write up and follow from another perspective because of the writability and readability that is included when using Go. While Go is a relatively new language, it has much interest from people so a tool has been created called GoMobile which compiles and runs pre-created Go applications for Android and iOS mobile applications. (OS and Android Programming with Go.)

Go is a very exciting language that has entered the market not very long ago. The first thing that attracted me to Go was that it was created, developed, and still maintained at Google. Without digging deep into it I predicted that the language would be produced and created with quality because it is Google’s premier choice of what language they use and optimized for the projects they work on. The number one thing that concerned me at first is that Go would have been developed and created for expert programmers who had training and instructions to people that were close to the creator. As it turns out, Go is created so efficiently for readability and writability that it is very easy to learn and teach. I would like to see Go someday being taught as a primary language over Java or Python because it is that easy to use, much easier to teach and learn.

Citations

“Documentation.” *Go*, golang.org/doc/.

Editor, QArea News. “Golang: How It Started - A History of Success - QArea Blog.” *Software Development Company*, 7 Oct. 2019, qarea.com/blog/the-evolution-of-go-a- history-of-success.

Griesemer , Robert. “The Evolution of Go.” Google Inc., 2015.

Hiwarale, Uday. “Anatomy of Channels in Go - Concurrency in Go.” *Medium*, RunGo, 10 Sept. 2019, medium.com/rungo/anatomy-of-channels-in-go-concurrency-in-go- 1ec336086adb.

“IOS and Android Programming with Go.” *SitePoint*, 8 Dec. 2015, [www.sitepoint.com/ios-and-android-programming-with-go/](http://www.sitepoint.com/ios-and-android-programming-with-go/).

“Memory Blocks.” *Memory Blocks - Go 101 (an Online Book for Go Programming Language + Golang Knowledge Base)*, go101.org/article/memory-block.html.

Work, Anshul\_AggarwalBelieves in Smart, et al. “Scope of Variables in Go.” *GeeksforGeeks*, 22 July 2019, [www.geeksforgeeks.org/scope-of-variables-in-go/.](http://www.geeksforgeeks.org/scope-of-variables-in-go/.(3))

Roussel, Achille, and Rick Branson. “Allocation Efficiency in High-Performance Go Services.” *Segment*, segment.com/blog/allocation-efficiency-in-high-performance-go-services/.

Rough Draft vs. Final Draft

After receiving my rough draft back I knew I had some changes to make and these bullet points identify the changes.

* Grammar errors
* Citations in bibliography are more scholarly and cited within the paper
* Spoke more of Channels and Goroutines and how they make Go a unique language
* Talked more over swap function in 1st examples
* Changed 2nd example to use channels
* Expanded more on readability and writability
* Clarified confusion with things that happened at compile and run time
* Added section over parameter passing and functions