# GoLang

GoLang is a new programming language developed by Google. It’s meant to be extremely quick and be a replacement for C, Except a bit easier. According to Keval Patel[[1]](#footnote-1): Moore’s Law is failing, clock speed is not nearly increasing as much as expected. So because of that, manufacturers have begun adding more and more cores. Google’s answer to this problem is Go. Most modern programming languages are single threaded environments Go uses ‘goroutines’ has grow-able segmented stacks. This means they will use more memory only when needed. Goroutines have a faster startup than threads. They come with built-in primitives to communicate safely between themselves. Go compiles with C++ which is extremely fast. Go uses garbage collection to allocation and removal of objects. Go is easy to maintain according Keval. There aren’t any classes, and it didn’t doesn’t have inheritance, or constructors or generics. Go is more efficient and faster than java, and almost as quick as c++. This is a win-win for both humans and processors. Go’s syntax is very stable and hasn’t changed since its implementation in 2012.

An example of Go’s async api is posted below.[[2]](#footnote-2)

type fetchResult struct {

Message string

Error error

}

func fetchA() fetchResult {

time.Sleep(time.Second \* 4)

return fetchResult{"A data", nil}

}

The syntax is simple and extremely easy to read. You can tell that when fetchA is called, it sleeps for 4 seconds then returns “A data”.

Another example of async is: [[3]](#footnote-3)

func fetchSequential() {

a := fetchA()

fmt.Println(a)

b := fetchB()

fmt.Println(b)

c := fetchC()

fmt.Println(c)

}

func fetchA() fetchResult {

fmt.Println("fetching A")

time.Sleep(time.Second \* 4)

return fetchResult{"A data", nil}

}

func fetchB() fetchResult {

fmt.Println("fetching B")

time.Sleep(time.Second \* 2)

return fetchResult{"B data", nil}

}

func fetchC() fetchResult {

fmt.Println("fetching C")

time.Sleep(time.Second \* 1)

return fetchResult{"C data", nil}

}

func main() {

fetchConcurrent()

// fetchSequential()

}

package main

import (

"fmt"

"time"

)

type fetchResult struct {

Message string

Error error

}

func fetchConcurrent() {

aChan := make(chan fetchResult, 0)

go func(c chan fetchResult) {

c <- fetchA()

}(aChan)

bChan := make(chan fetchResult, 0)

go func(c chan fetchResult) {

c <- fetchB()

}(bChan)

cChan := make(chan fetchResult, 0)

go func(c chan fetchResult) {

c <- fetchC()

}(cChan)

// order doesn't really matter!

a := <-aChan

b := <-bChan

c := <-cChan

fmt.Println(a)

fmt.Println(b)

fmt.Println(c)

}

This returns with:

*fetching C*

*fetching A*

*fetching B*

*{A data <nil>}*

*{B data <nil>}*

*{C data <nil>}*

*// Commenting text in Go uses /\* block comments \*/ and line comments just like most*

*// modern languages.*

For loops do look a little different from most other popular high level languages, except for Ada of course.

for i := 0; i < 10; i++ {

fmt.Println(i)

}

func hello() (string, string) {

return "hello", "world"

}

func main() {

a, b := hello()

fmt.Println(a, b)

}

Go functions can also return multiple values. Other high level languages, you’ve got to return a list or an object or like that to get the return values.

Go is open sourced and has existed for 8 years now. Its backed by Google and other large companies to keep it moving and always working correctly. It is an extremely quick and performant language which lets developers really build out the program efficiently.

1. <https://medium.com/@kevalpatel2106/why-should-you-learn-go-f607681fad65> [↑](#footnote-ref-1)
2. <https://github.com/pazams/go-for-javascript-developers#preface> [↑](#footnote-ref-2)
3. https://play.golang.org/p/2BVwtos4-j [↑](#footnote-ref-3)