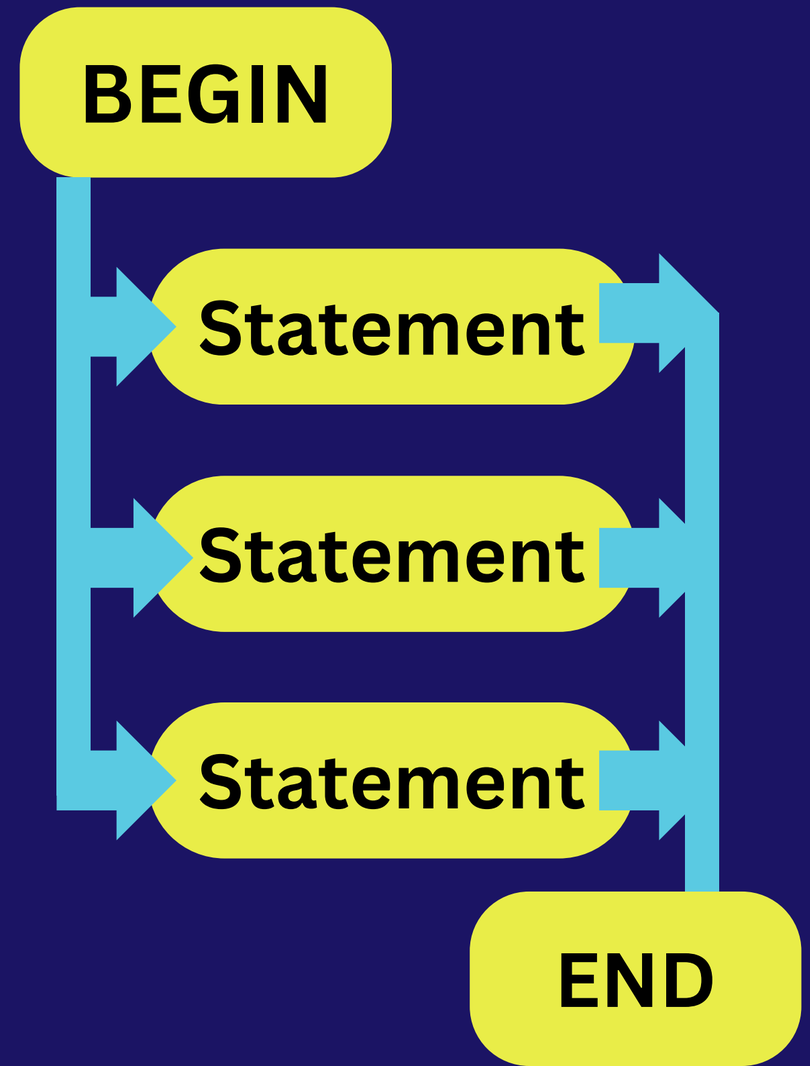
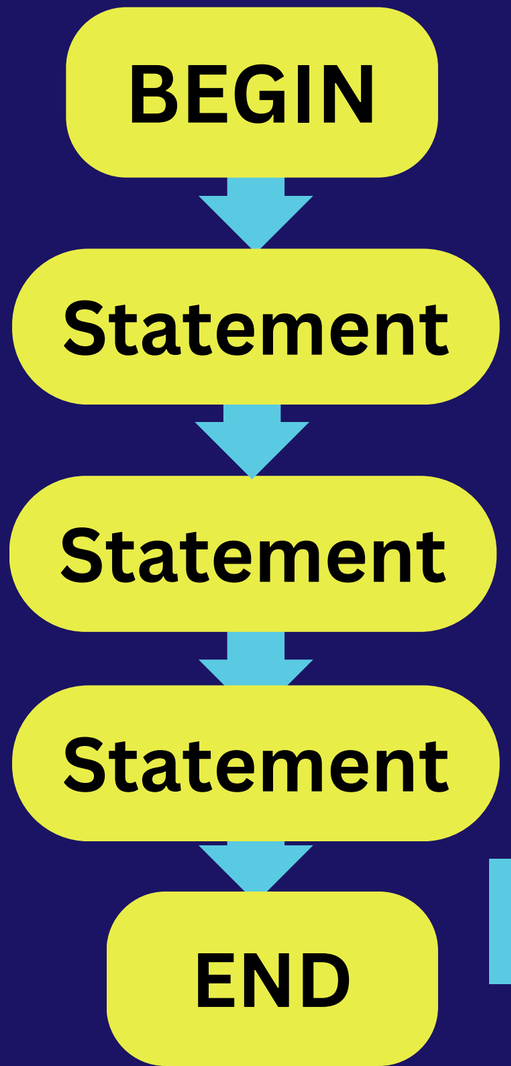
Concurrency

What is Concurrency?

Concept of executing multiple tasks simultanenously by utilizing all available resources more effectively

Concurrent v/s Sequential





Time Difference

- 2 Go Program which fetches the weather of provided cities through JSON response from API, one sequential and other concurrent can be useful for noticing the Time Difference.

Weather for Surat is 299.14 K

Weather for Kolkata is 300.12 K

Weather for Kharagpur is 300.16 K

Weather for Mumbai is 301.14 K

Execution time: 473.975608ms

Sequential

Weather for Surat: 299.14 K

Weather for Kolkata: 300.12 K

Weather for Kharagpur: 300.16 K

Weather for Mumbai: 301.14 K

Execution time: 316.347076ms

Concurrent

Benefits of Concurrency

- Running of multiple applications
- Better resource utilization
- Better average response time
- Better performance

Concurrent Programming is done in Go

Through :

- Goroutines
- Channels
- Waitgroups
- Mutexes

Goroutines

- A Goroutine is a Function or method which executes independently and simultaneously in connection with any other Goroutines present in your program.
- Main Function itself is a Goroutine.
- Goroutines can communicate using the channels.

What is Channels? We will see it Later!

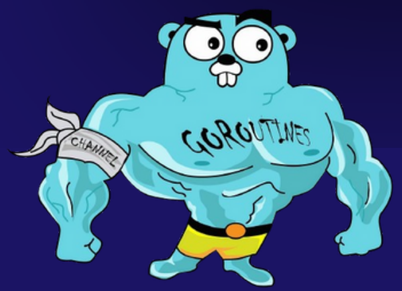
Declaration of Goroutine

```go
func name(){
// statements
}

// using go keyword as the
// prefix of your function call
go name()
```

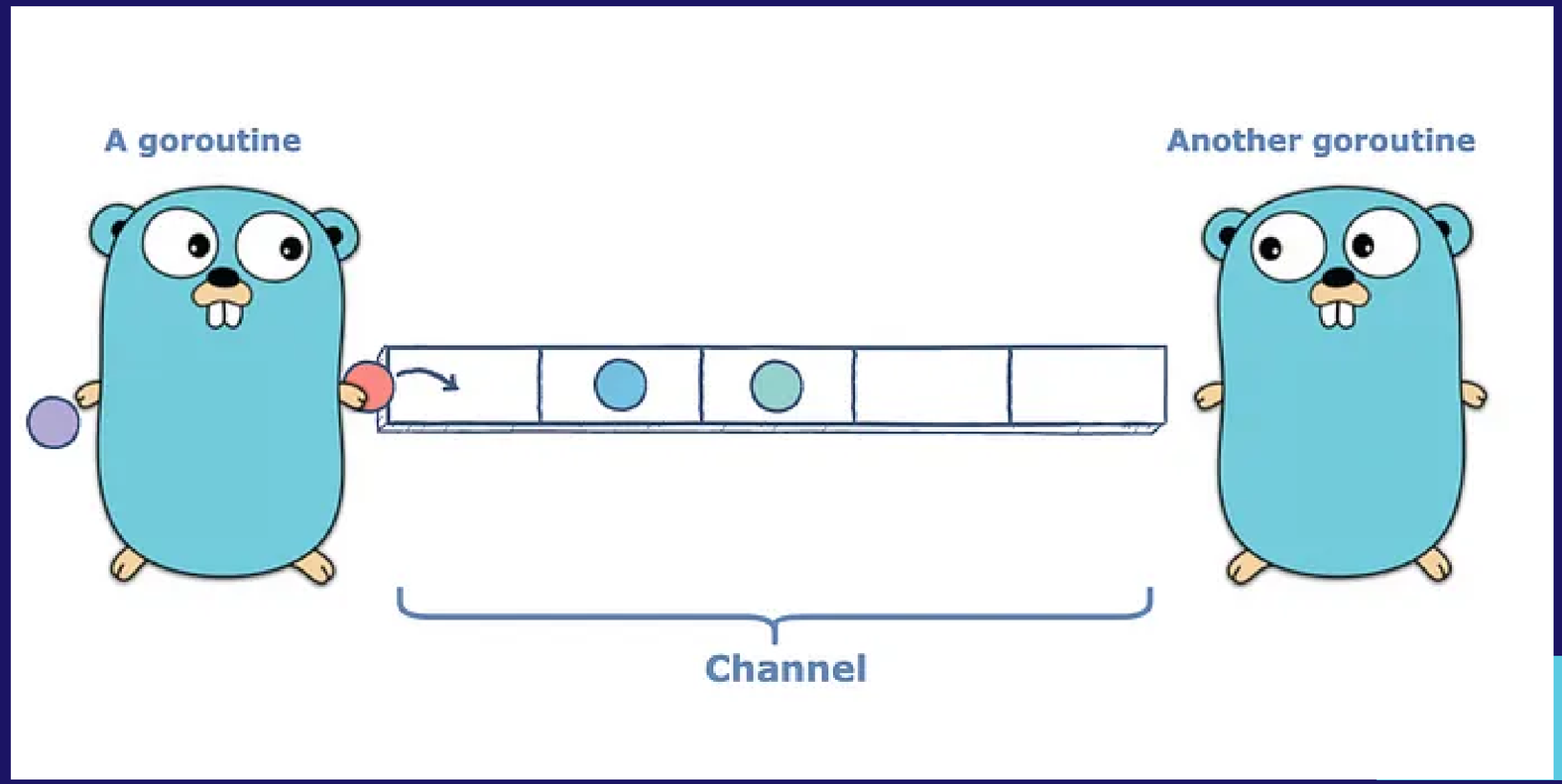
```go
//Anonymous Function call

go func (parameter_list){
  //statements
} (arguments)
```



Channels

- A channel is a medium through which a goroutine communicates with another goroutine.



Working with Channels

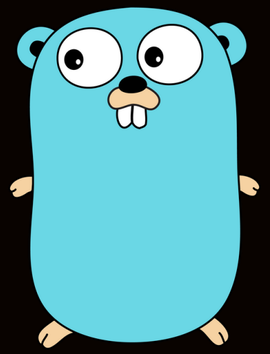
```
channel_name:= make(chan Type)
//Type can be int, string, char, float64 etc..

Mychannel <- element //For Sending Element into Channel.

element := <- Mychannel //For Receiving Element from a Channel.

close() //For Closing a Channel

ele, ok:= <- Mychannel //if value of ok is true then channel is open else closed.
```



Waitgroup

- A WaitGroup is a Go struct used to wait for a collection of goroutines to finish.

Why we need Waitgroup??

Let’s See Through a Live Example!

```
//Some Commands for working with Waitgroup.

var wg sync.WaitGroup //creates a Waitgroup

wg := new(sync.Waitgroup) //creates a Waitgroup Pointer

wg.Add(2) //adds 2 to the counter

defer wg.Done() //decrement 1 in counter

wg.Wait() //wait till all goroutines finish --i.e. counter becomes zero.
```

Mutex {Mutual Exclusion}

- A Mutex is a method used as a locking mechanism to ensure that only one Goroutine is accessing the critical section of code at any point of time

Why we need Mutex??

Let’s See Through a Live Example Again!

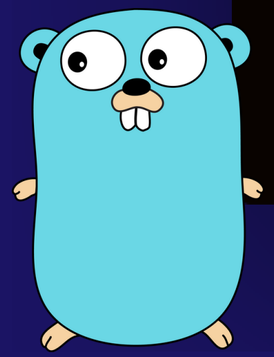
```
//Some Commands for working with Mutex {Mutual Exclusion}.

m := new(sync.Mutex) //creates a Mutex Pointer

var m sync.Mutex //creates a Mutex

m.Lock() //for Locking the next segment of code for Goroutines

m.Unlock() //For Unlocking
```



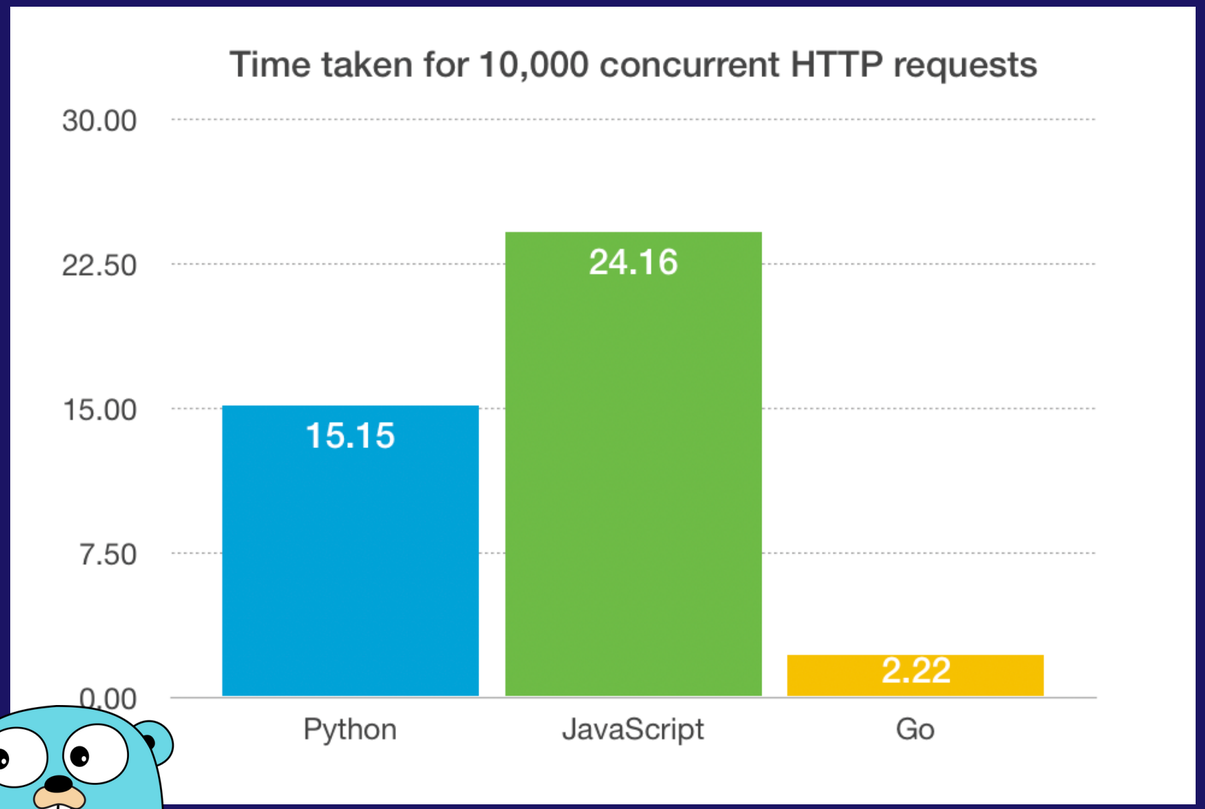
How it is Different from Python?

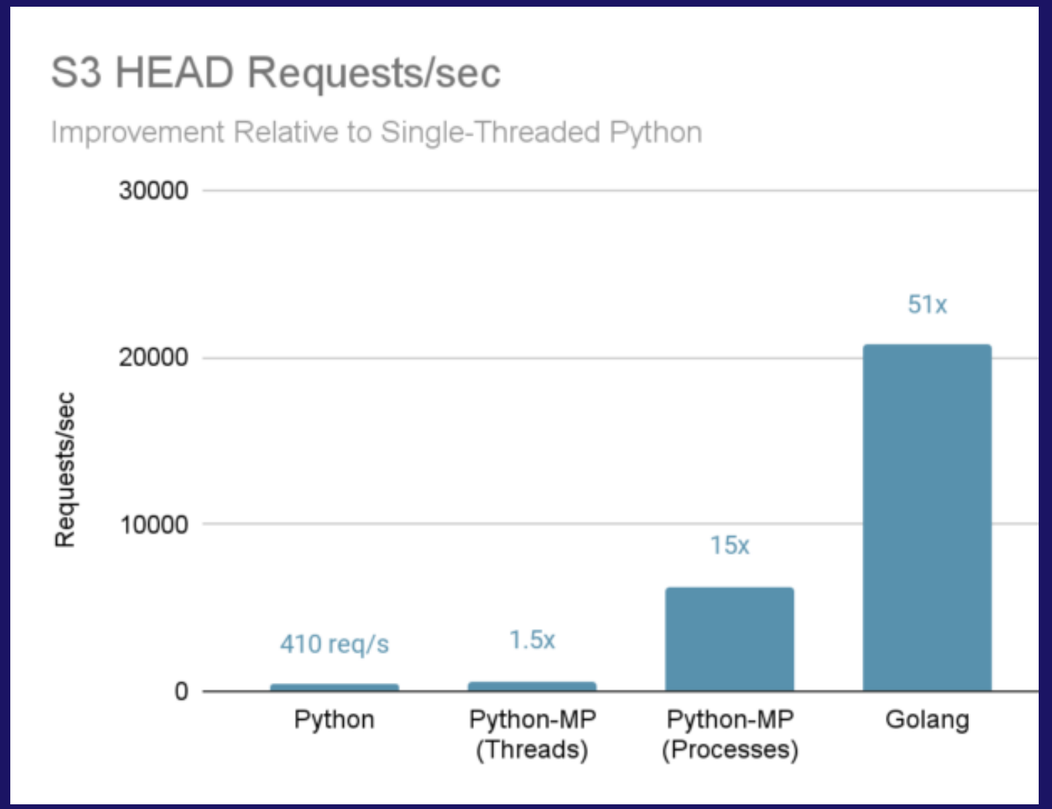
- Python uses Global interpreter Lock (GIL). it’s a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.
- This means that only one thread can be in a state of execution at any point in time.
- Go’s goroutines provide a lightweight and efficient concurrency model.

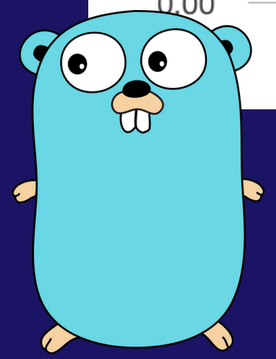
How it is Different from Python?

- Provided Go is way much Faster in terms of Concurrency and nowadays Modern Software Development mainly depends on Using Maximum Resources, Go Language has much Higher Advantage in terms of Concurrency.

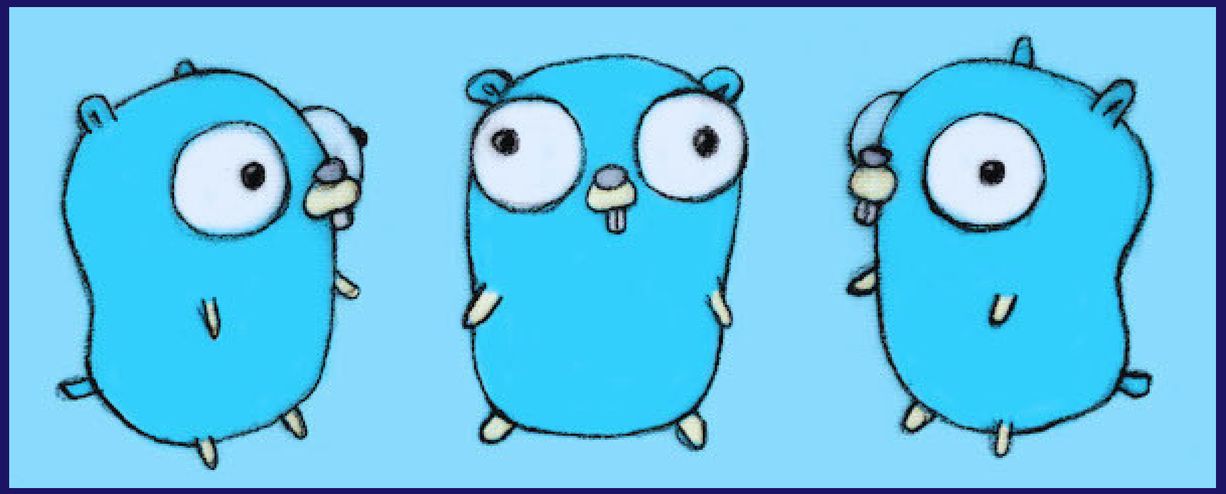
Concurrency Time & Resources Comparison







Q/A Session



Thank You !