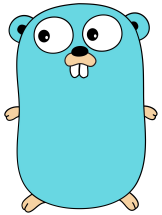
Go from scratch: Beginners-Friendly Guide

by Denis Shchuka

Parallelism & Concurrency



Basic Concepts

- Concurrency allows the application to do more than one thing at the same time by switching between working threads
- Parallelism allows to do them in the same time - in parallel.

PARALLELISM ≠ **CONCURRENCY**

Goroutine - lightweight thread of execution inside of a Go process. Goroutines work in parallel mode.

In Go goroutine can communicate with each other via **Channels**

Goroutines

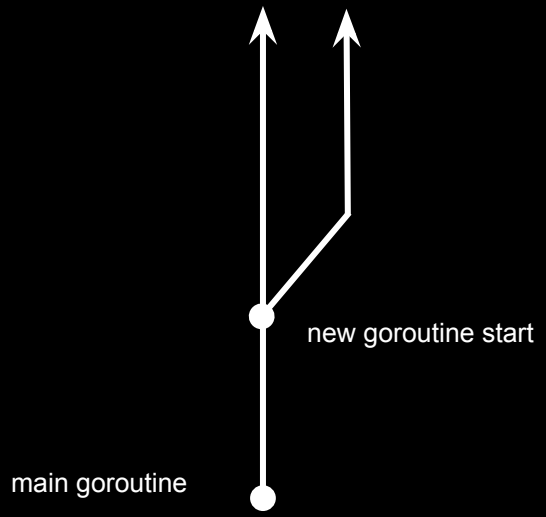
Goroutine is function that returns control to execution flow right after it was ran

```go
package main

import (
	"fmt"
	"time"
)

func main() {
	go calc_f()
	for i := 'a'; i <= 'z'; i++ {
		time.Sleep(500 * time.Millisecond)
		fmt.Printf("%c", i)
	}
}

func calc_f() {
	for i := 0; i < 100  i++ {
		fmt.Print(i)
		time Sleep(500 * time.Millisecond)
	}
}
```



Channels

Channel are the pipes that allow to pass data between goroutines

```
//declaration of new channel of  chan string data type
//allows to pass strings between goroutines
ch := make(chan string)

//send new value to channel
ch <- "hello!"

//read value from channel
msg <- ch

//close channel
close(ch)
```

Channels

TYPES OF CHANNELS

UNBUFFERED

```
ch := make(chan string)
```

- *synchronous communication*
- *the sender blocks until the receiver has received the value*

BUFFERED

```
ch := make(chan string, 10)
```

- *asynchronous communication*
- *the sender continues to execute after sending value to buffer*
- *if buffer is empty than receiver will be blocked waiting for putting new element by sender*