Go by Example: WaitGroups

To wait for multiple goroutines to finish, we can use a *wait group*.

This is the function we'll run in every goroutine.

Sleep to simulate an expensive task.

This WaitGroup is used to wait for all the goroutines launched here to finish. Note: if a WaitGroup is explicitly passed into functions, it should be done *by pointer*.

Launch several goroutines and increment the WaitGroup counter for each.

Avoid re-use of the same `i` value in each goroutine closure. See the FAQ for more details.

```
package main

import (
    "fmt"
    "sync"
    "time"
)

func worker(id int) {
    fmt.Printf("Worker %d starting\n", id)

    time.Sleep(time.Second)
    fmt.Printf("Worker %d done\n", id)
}

func main() {

    var wg sync.WaitGroup



    for i := 1; i <= 5; i++ {
        wg.Add(1)

        i := i
```

Wrap the worker call in a closure that makes sure to tell the WaitGroup that this worker is done. This way the worker itself does not have to be aware of the concurrency primitives involved in its execution.

Block until the WaitGroup counter goes back to 0; all the workers notified they're done.

Note that this approach has no straightforward way to propagate errors from workers. For more advanced use cases, consider using the errgroup package.

```
        go func() {
            defer wg.Done()
            worker(i)
        }()
    }

    wg.Wait()


}
```

```
$ go run waitgroups.go
Worker 5 starting
Worker 3 starting
Worker 4 starting
Worker 1 starting
Worker 2 starting
Worker 4 done
Worker 1 done
Worker 2 done
Worker 5 done
Worker 3 done
```

The order of workers starting up and finishing is likely to be different for each invocation.

Next example: Rate Limiting.