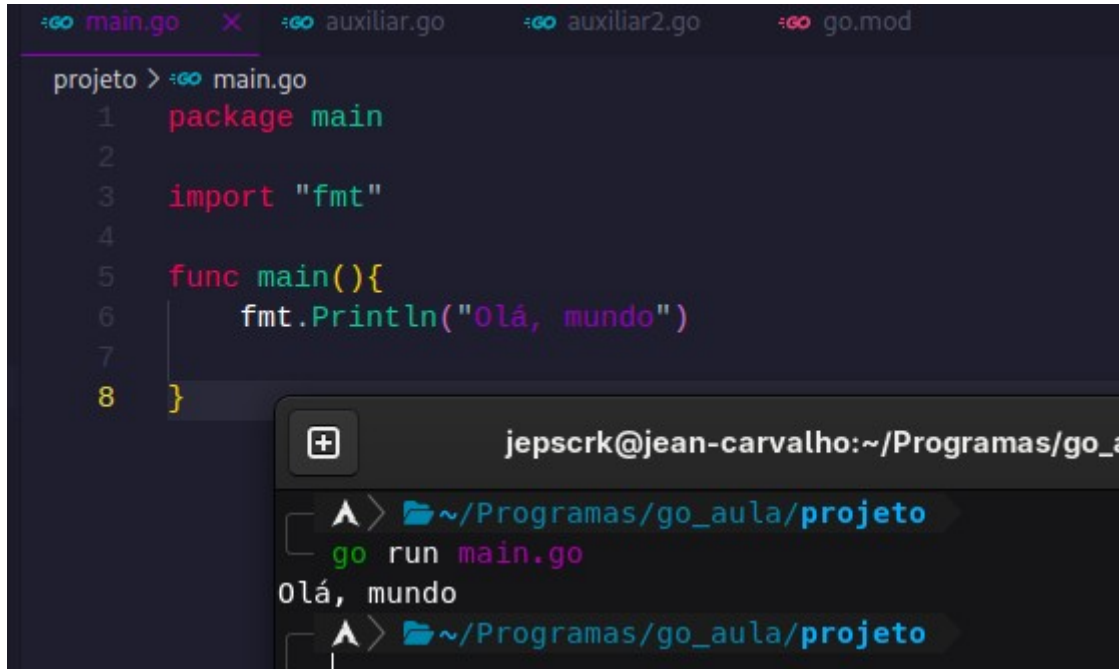


## “Hello, World!” em GO

Primeiro, criamos um projeto em GO, precisamos criar um modulo para que ele possa compilar o seu código. Para criar, basta digitar: **GO MOD INIT <nome do projeto>**. Feito isso, podemos dar o nosso primeiro print



```
main.go x auxiliar.go auxiliar2.go go.mod
projeto > go main.go
1 package main
2
3 import "fmt"
4
5 func main(){
6     fmt.Println("Olá, mundo")
7 }
8

jepsck@jean-carvalho:~/Programas/go_a...
^> ~/Programas/go_aula/projeto
go run main.go
Olá, mundo
^> ~/Programas/go_aula/projeto
```

No Go (Golang), `fmt` não é uma **função**, mas sim um **pacote da biblioteca padrão**. Ele é amplamente usado para **formatação e impressão de texto**, seja no terminal, em strings, arquivos, etc.



O que é o `fmt`?

`fmt` significa *format*, e o pacote fornece funções para formatar e imprimir dados.

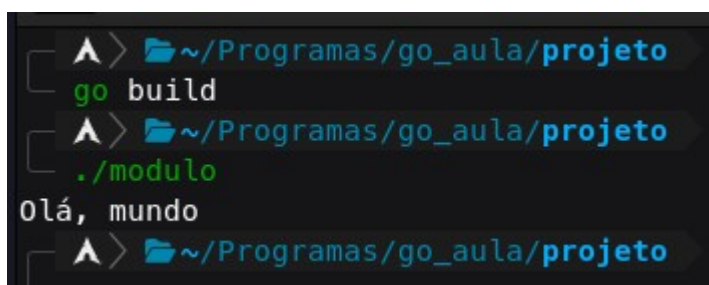
Veja algumas funções:

`fmt.Print()` Imprime na tela sem quebra de linha.

`fmt.Println()` Imprime na tela com quebra de linha.

`fmt.Printf()` Imprime com formatação (como `%d`, `%s`, etc.).

→ Como você viu na saída do terminal, executamos o nosso código, mas também podemos compilar em versão binário(ex: `.exe`, dependendo do seu SO) com o comando: **go build**



```
^> ~/Programas/go_aula/projeto
go build
^> ~/Programas/go_aula/projeto
./modulo
Olá, mundo
^> ~/Programas/go_aula/projeto
```

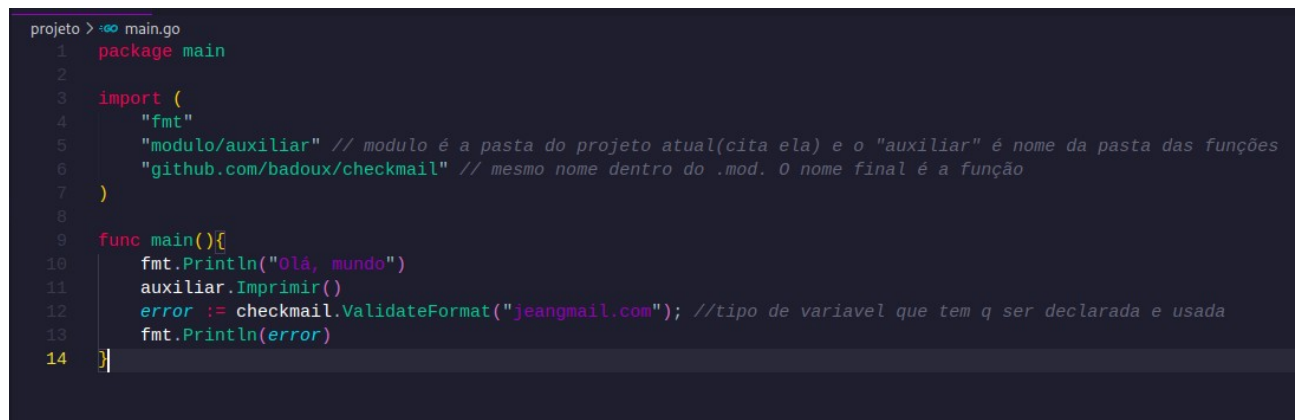
Mas dessa maneira, toda vez que você fizer uma atualização, você sempre terá que compilar para versão binária

## Pacotes

Vamos supor que você criou uma função e quer importar para o seu arquivo main. Vamos criar uma pasta dentro do nosso modulo para guardar as funções e iremos criar apenas para imprimir os dados na tela:



```
projeto > auxiliar > funcao.go
1 package auxiliar
2
3 import "fmt"
4
5 func Imprimir(){
6     fmt.Println("Olá");
7
8 }
```



```
projeto > main.go
1 package main
2
3 import (
4     "fmt"
5     "modulo/auxiliar" // modulo é a pasta do projeto atual(cita ela) e o "auxiliar" é nome da pasta das funções
6     "github.com/badoux/checkmail" // mesmo nome dentro do .mod. O nome final é a função
7 )
8
9 func main(){
10     fmt.Println("Olá, mundo")
11     auxiliar.Imprimir()
12     error := checkmail.ValidateFormat("jeangmail.com"); //tipo de variavel que tem q ser declarada e usada
13     fmt.Println(error)
14 }
```

Não temos que especificar o nome do arquivo. O Go organiza o código em **pacotes (packages)**, e **cada pasta representa um pacote**. Quando você faz: `import "modulo/auxiliar"`

Você está dizendo: *"Importe o pacote chamado auxiliar que está dentro da pasta modulo."* O compilador do Go então **procura todos os arquivos .go dentro da pasta modulo/auxiliar/** e compila todos juntos, desde que tenham `package auxiliar` no topo.

Você **não precisa** (nem consegue) fazer algo como `auxiliar.auxiliar.Imprimir()` ou citar o nome do arquivo `.go`, porque o Go **agrupa todos os arquivos do mesmo pacote como um só conjunto**.

Mas quando você tem outro arquivo `.go` dentro da pasta(pacote) `auxiliar`, você não precisa especificar a pasta, ele já compila tudo junto o que está dentro do pacote

```
main.go  auxiliar.go  auxiliar2.go x
projeto > auxiliar > auxiliar2.go
1 package auxiliar
2
3 import "fmt"
4
5 func escrever(){
6     fmt.Println("Escrevendo")
7 }
```

```
main.go  auxiliar.go x  auxiliar2.go
projeto > auxiliar > auxiliar.go
1 package auxiliar
2
3 import "fmt"
4
5 func Imprimir(){
6     fmt.Println("ola");
7     escrever()
8 }
```

detalhe: se for importada, tem que ser usada, se não, dar erro

## 📌 Por quê?

Porque o Go trata a pasta como um único pacote.

Todos os arquivos .go daquela pasta, desde que comecem com:

```
package auxiliar
```

vão ser agrupados como se fossem um só bloco de código. Isso é parte da simplicidade e organização que o Go busca.

Então, e o main... ?

Em Go, package main é o ponto de entrada de um programa executável. Ou seja:

- **package main** indica que esse arquivo pertence a um programa que pode ser compilado e executado diretamente (go run, go build, etc).
- Ele precisa obrigatoriamente ter uma função **main()**, que é o que o Go vai rodar primeiro.

## Variáveis

Vamos primeiro ver como podemos declarar variáveis, que podem ser feitas diversas formas

```
projeto > main.go
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main(){
8     var simples1 = "Simples 1"
9     fmt.Println(simples1)
10    const constante = "variavel que não muda"
11    fmt.Println(constante)
12    nmuda := "Foi declarada ? tem que ser usada"
13    fmt.Println(nmuda)
14 }
```

```
jepscrk@jean-carvalho:~/Programas/go_aula/p
  ▲> ~/Programas/go_aula/projeto
  go run main.go
Simples 1
variavel que não muda
Foi declarada ? tem que ser usada
  ▲> ~/Programas/go_aula/projeto
```

Não precisamos especificar o tipo da variável, mas podemos e é recomendado para deixar o código mais explícito

```
7 func main(){
8     var simples1 string = "Simples 1"
9     fmt.Println(simples1)
10    const num int = 10
11    const ponto float64 = 1.5
12    fmt.Println(num, ponto)
13    var boleano bool = true
14    fmt.Println(boleano)
15    nmuda := "Foi declarada ? tem que ser usada"
16    fmt.Println(nmuda)
17 }
```