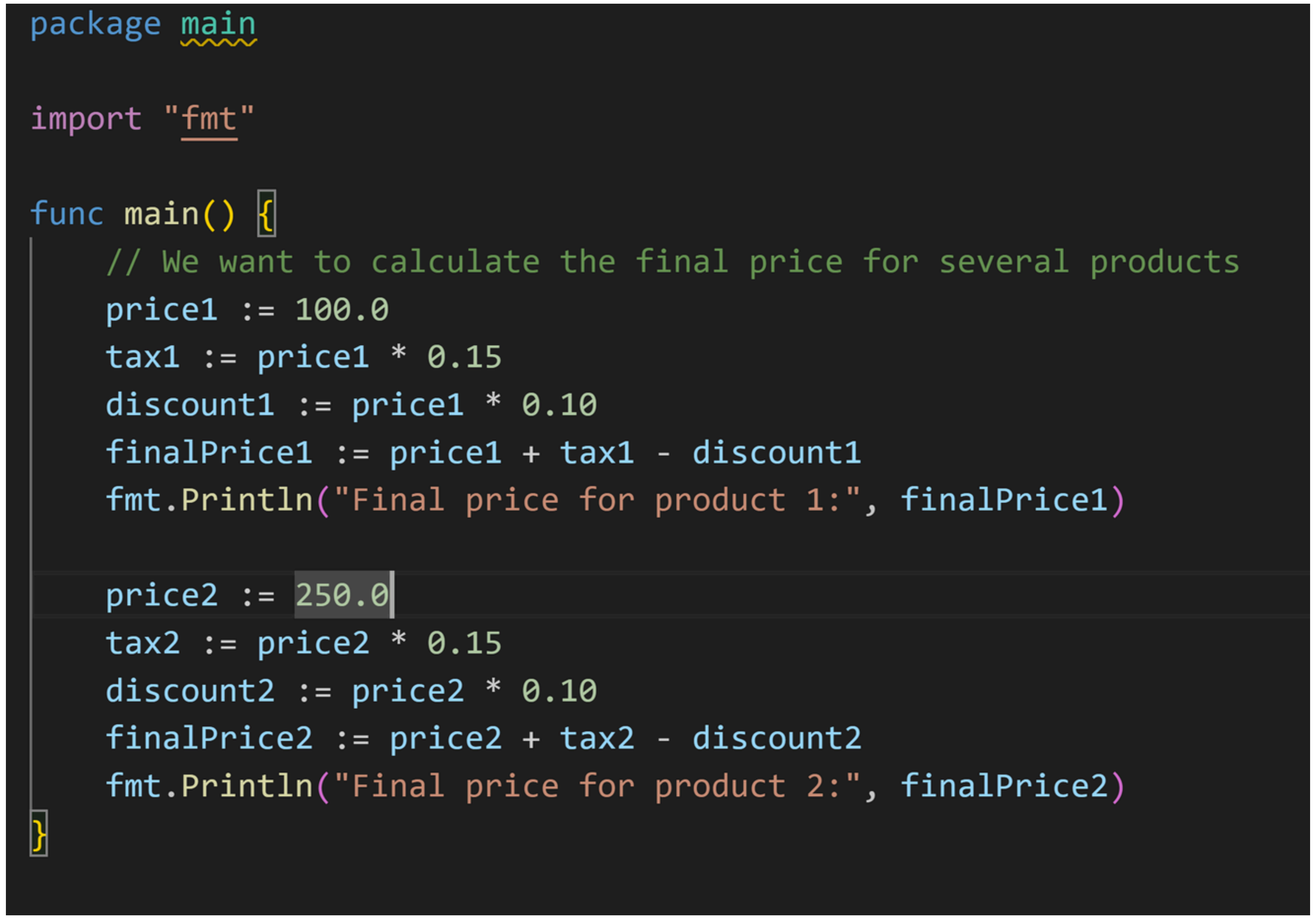
Functions

What are Functions?

A function is **a block of reusable code** that performs a specific task.

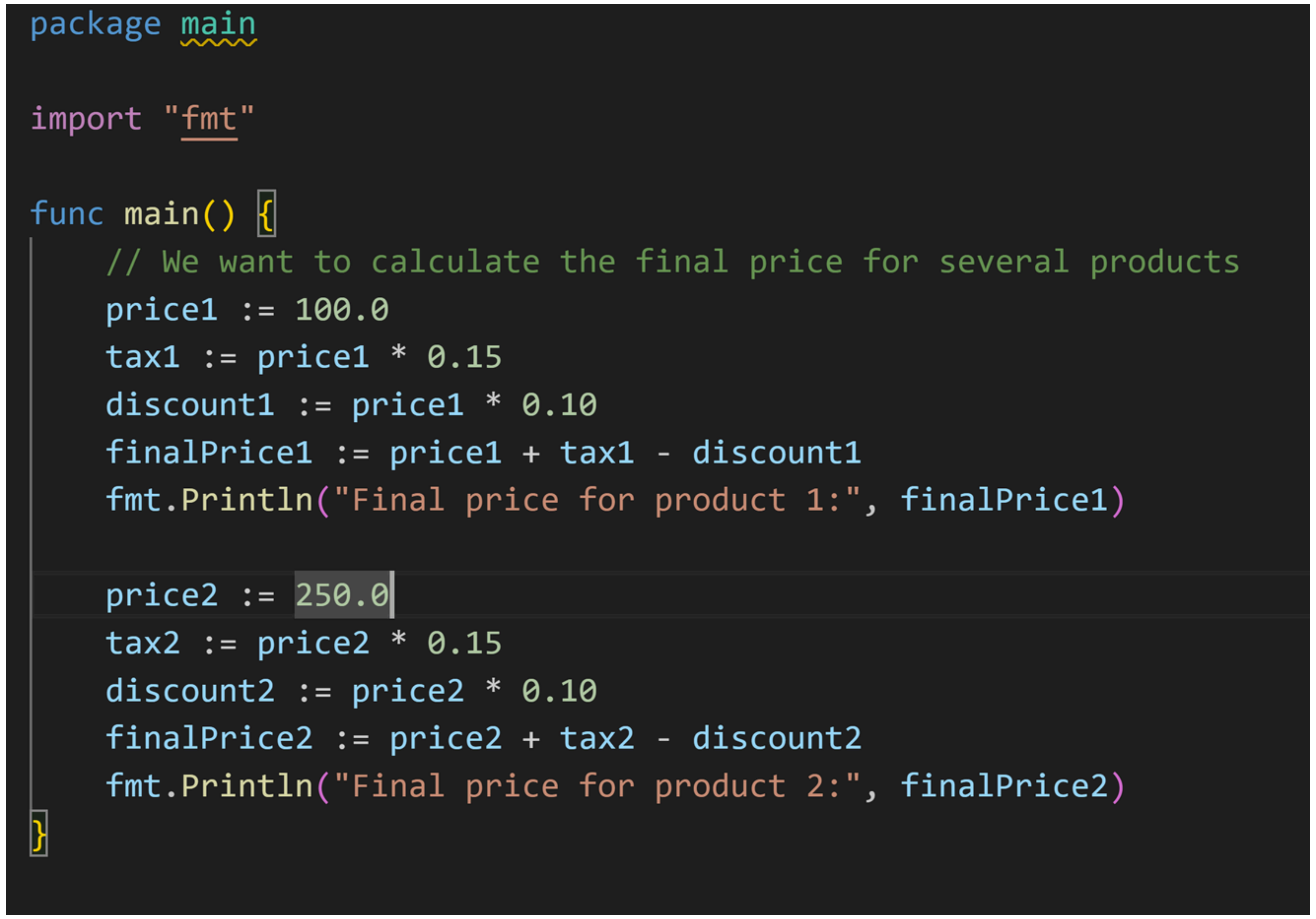
Instead of writing the **same logic multiple times,** we can package it into a function and **call it whenever needed.**

What are Functions?

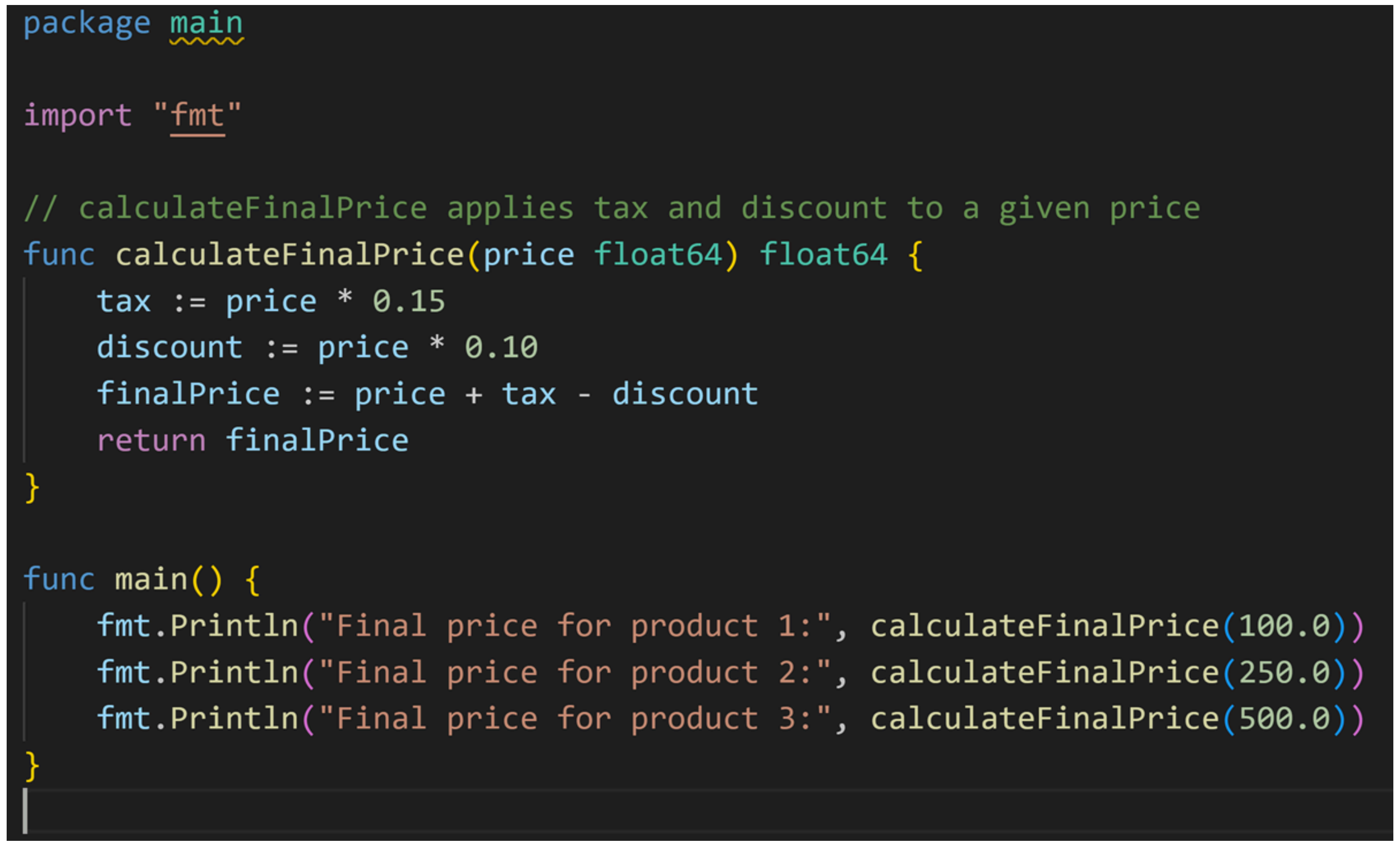


```go
package main

import "fmt"

func main() {
	// We want to calculate the final price for several products
	price1 := 100.0
	tax1 := price1 * 0.15
	discount1 := price1 * 0.10
	finalPrice1 := price1 + tax1 - discount1
	fmt.Println("Final price for product 1:", finalPrice1)

	price2 := 250.0
	tax2 := price2 * 0.15
	discount2 := price2 * 0.10
	finalPrice2 := price2 + tax2 - discount2
	fmt.Println("Final price for product 2:", finalPrice2)
}
```

What are Functions?



```go
package main

import "fmt"

func main() {
    // We want to calculate the final price for several products
    price1 := 100.0
    tax1 := price1 * 0.15
    discount1 := price1 * 0.10
    finalPrice1 := price1 + tax1 - discount1
    fmt.Println("Final price for product 1:", finalPrice1)

    price2 := 250.0
    tax2 := price2 * 0.15
    discount2 := price2 * 0.10
    finalPrice2 := price2 + tax2 - discount2
    fmt.Println("Final price for product 2:", finalPrice2)
}
```

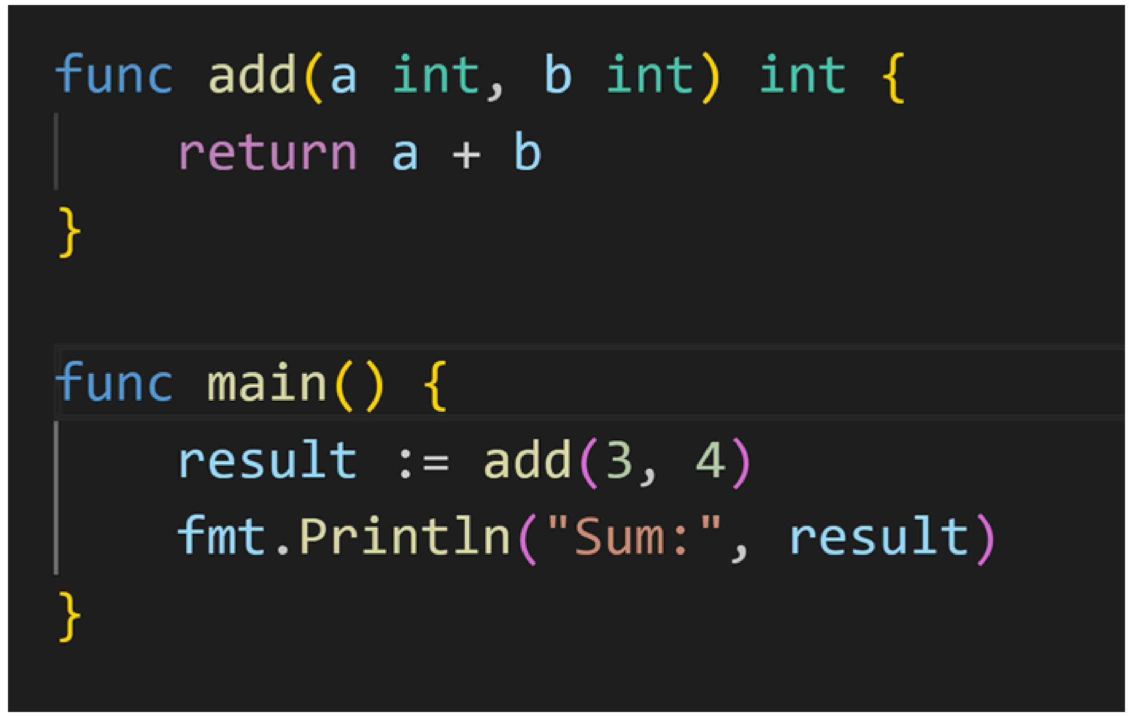


```go
package main

import "fmt"

// calculateFinalPrice applies tax and discount to a given price
func calculateFinalPrice(price float64) float64 {
    tax := price * 0.15
    discount := price * 0.10
    finalPrice := price + tax - discount
    return finalPrice
}

func main() {
    fmt.Println("Final price for product 1:", calculateFinalPrice(100.0))
    fmt.Println("Final price for product 2:", calculateFinalPrice(250.0))
    fmt.Println("Final price for product 3:", calculateFinalPrice(500.0))
}
```

What are Functions?

```
func functionName(parameters) returnType {
    // function body
    return value
}
```

```
func functionName(parameters) returnType {
    // function body
    return value
}
```



```go
func add(a int, b int) int {
    return a + b
}

func main() {
    result := add(3, 4)
    fmt.Println("Sum:", result)
}
```

Why Do We Use Functions?

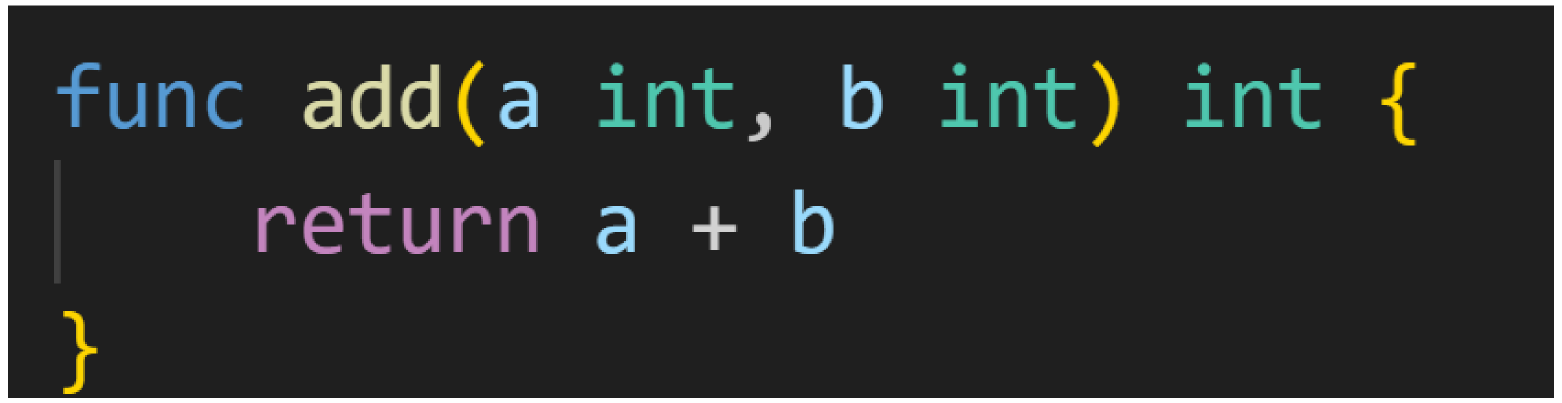
Reusability	Write once, use many times.
Readability	Break complex problems into smaller, manageable parts.
Maintainability	Easier to update code when logic is encapsulated in functions.
Testing	Small, focused functions are easier to test.

Functions

Variadic Functions

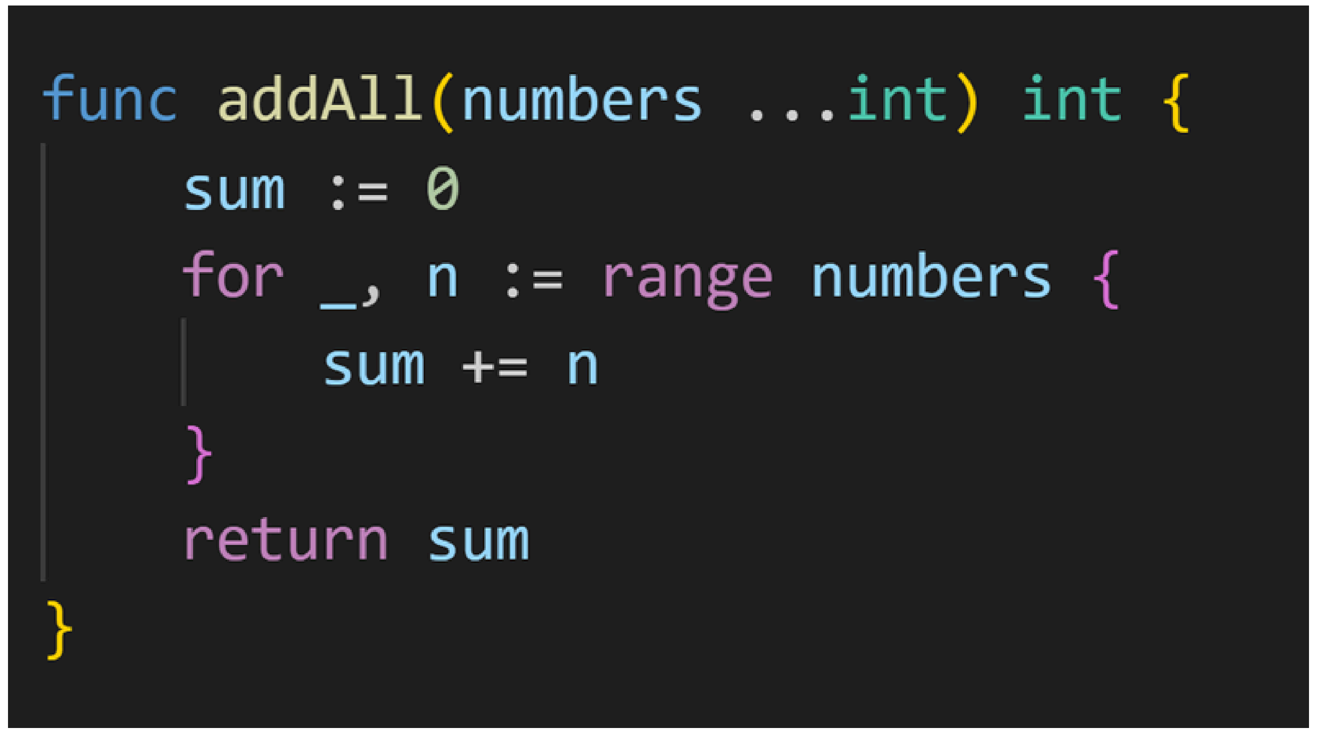
A variadic function is a function that can **accept zero or more arguments of the same type.**

Non-Variadic Function



```go
func add(a int, b int) int {
    return a + b
}
```

Variadic Function



```go
func addAll(numbers ...int) int {
    sum := 0
    for _, n := range numbers {
        sum += n
    }
    return sum
}
```

Functions

Anonymous Functions

In Go, **functions don't always need a name.**
You can declare **anonymous functions** (also called function literals) right inside your code.

when:

- You want a **short, one-off piece of functionality.**
- You **don't need to reuse** the function elsewhere.
- You want to pass a function as **an argument or return value.**

```
func(parameters) returnType {
    // function body
}
```