DWARVES FOUNDATION

Functions & Packages

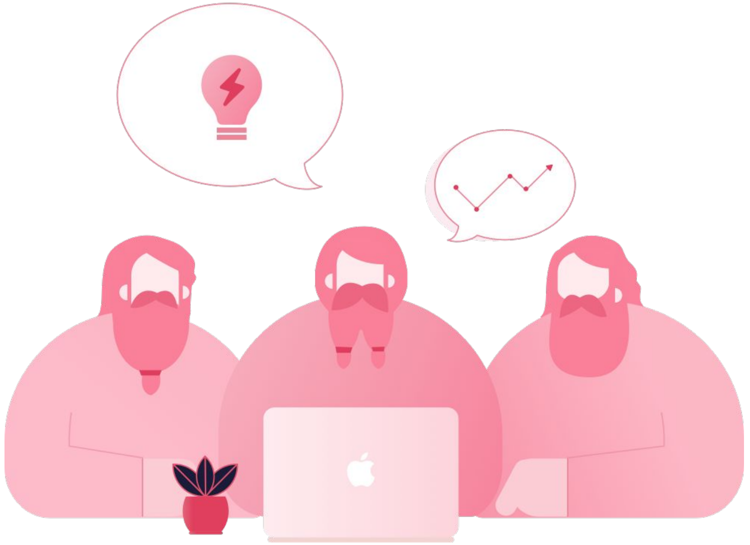
Understanding functions and working with packages

Author Name

Hieu Phan

@hieupq

andy@d.foundation



DWARVES FOUNDATION

Agenda

Understanding functions and working with packages

DWARVES FOUNDATION

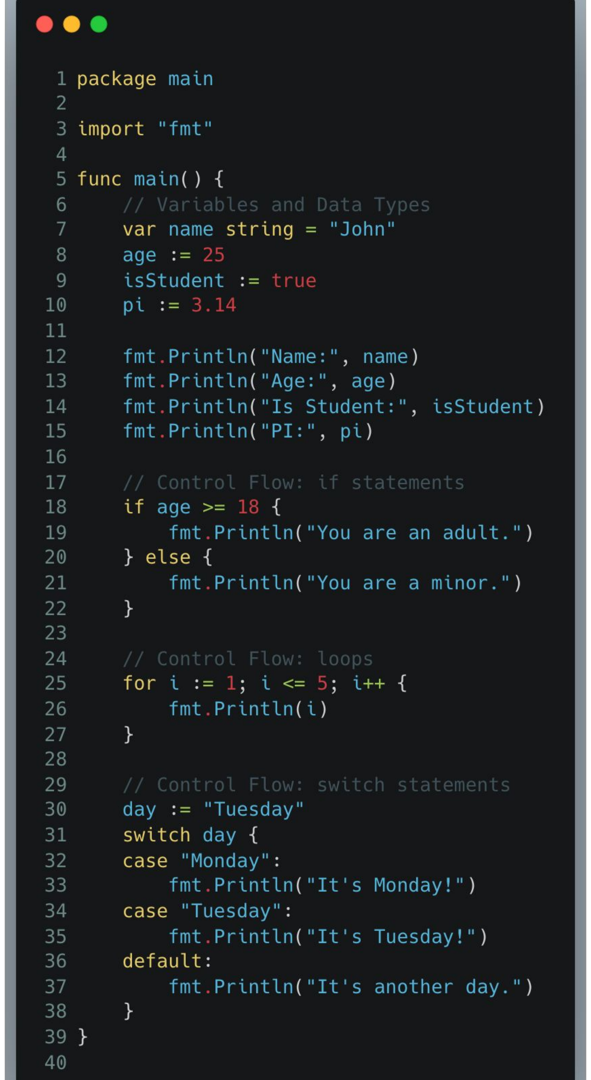
Introduction

Recap of Day 1 and Day 2 overview

DWARVES FOUNDATION

Recap of Day 1

- Variables and Data types
- Control Flow: if, switch
- Loops

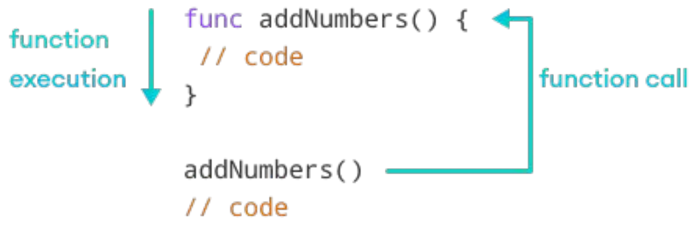


```go
package main

import "fmt"

func main() {
    // Variables and Data Types
    var name string = "John"
    age := 25
    isStudent := true
    pi := 3.14

    fmt.Println("Name:", name)
    fmt.Println("Age:", age)
    fmt.Println("Is Student:", isStudent)
    fmt.Println("PI:", pi)

    // Control Flow: if statements
    if age >= 18 {
        fmt.Println("You are an adult.")
    } else {
        fmt.Println("You are a minor.")
    }

    // Control Flow: loops
    for i := 1; i <= 5; i++ {
        fmt.Println(i)
    }

    // Control Flow: switch statements
    day := "Tuesday"
    switch day {
    case "Monday":
        fmt.Println("It's Monday!")
    case "Tuesday":
        fmt.Println("It's Tuesday!")
    default:
        fmt.Println("It's another day.")
    }
}

```

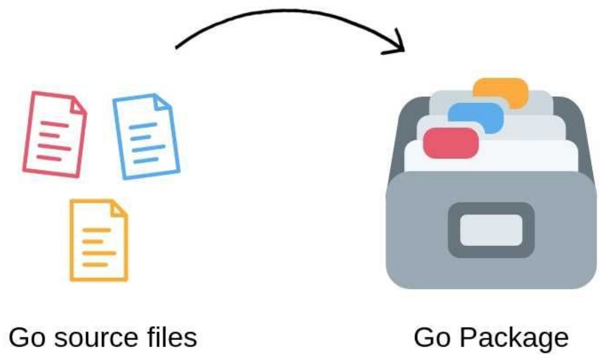
DWARVES FOUNDATION

Day 2

- Functions as a fundamental building block of Go programs.
- Packages as a mechanism for organizing and reusing code.



```
func addNumbers() {
 // code
}

addNumbers()
// code
```



DWARVES FOUNDATION

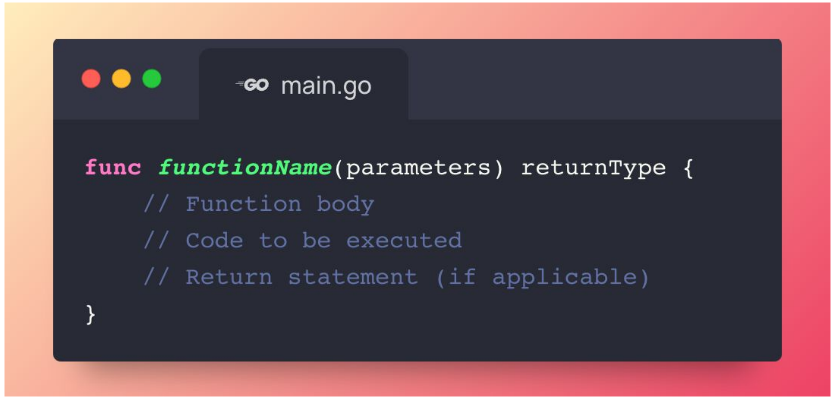
Function Basics

Explain the syntax of function declaration in Go

DWARVES FOUNDATION

Syntax

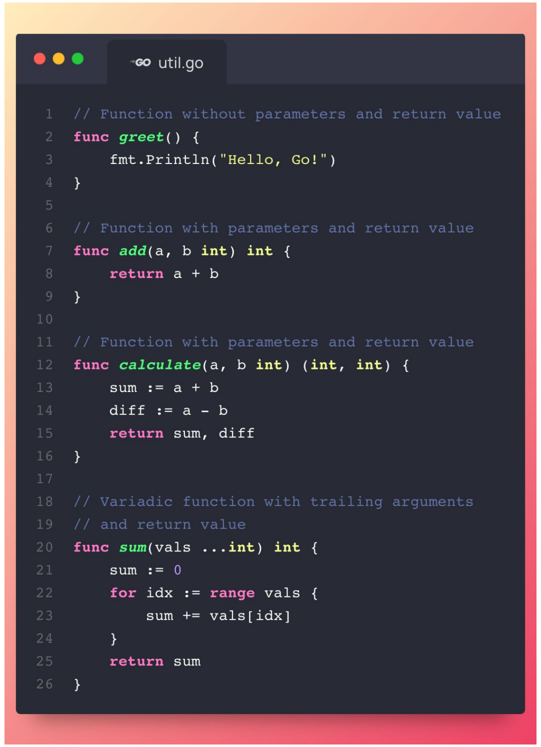
- **functionName** is the identifier for the function.
- **parameters** are optional and represent values passed to the function for processing.
- **returnType** is *optional* and specifies the type of value that the function returns.



```go
func functionName(parameters) returnType {
    // Function body
    // Code to be executed
    // Return statement (if applicable)
}
```

Syntax

- **functionName** choose a meaningful name that describes the purpose of the function.
- **parameters** multiple parameters can be defined, separated by commas. Each parameter has a name and a type
- **returnType** if the function doesn't return a value, the return type can be omitted.



```go
// Function without parameters and return value
func greet() {
    fmt.Println("Hello, Go!")
}

// Function with parameters and return value
func add(a, b int) int {
    return a + b
}

// Function with parameters and return value
func calculate(a, b int) (int, int) {
    sum := a + b
    diff := a - b
    return sum, diff
}

// Variadic function with trailing arguments
// and return value
func sum(vals ...int) int {
    sum := 0
    for idx := range vals {
        sum += vals[idx]
    }
    return sum
}
```

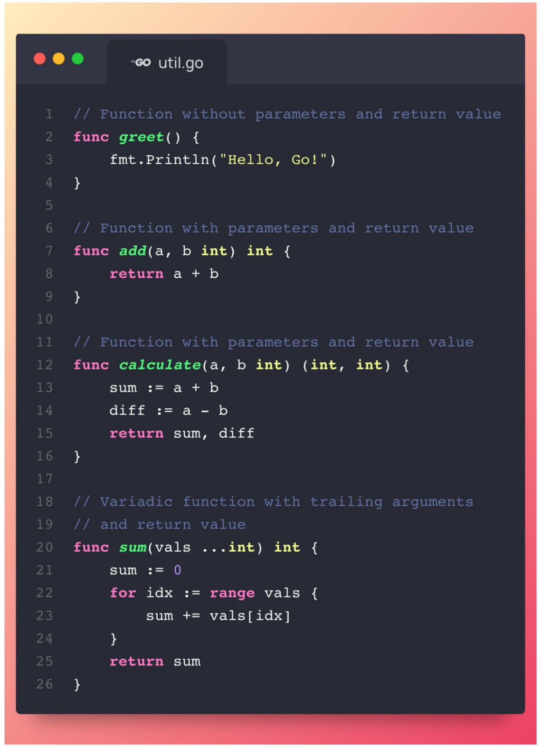
DWARVES FOUNDATION

Function Signature

The function signature is a combination of the function name, parameter list, and return type.

It defines the unique identity of a function within a package.

Overloading functions (same name, different parameter list) is **NOT** supported in Go.


util.go

```go
// Function without parameters and return value
func greet() {
    fmt.Println("Hello, Go!")
}

// Function with parameters and return value
func add(a, b int) int {
    return a + b
}

// Function with parameters and return value
func calculate(a, b int) (int, int) {
    sum := a + b
    diff := a - b
    return sum, diff
}

// Variadic function with trailing arguments
// and return value
func sum(vals ...int) int {
    sum := 0
    for idx := range vals {
        sum += vals[idx]
    }
    return sum
}
```

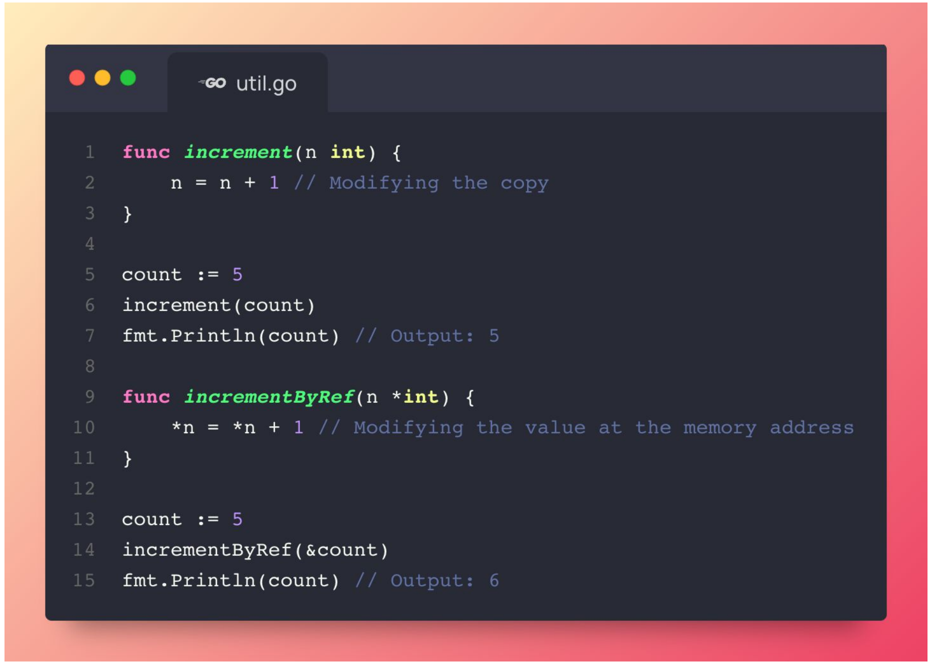
DWARVES FOUNDATION

Function Invocation

Function invocation is the process of executing a function in Go.

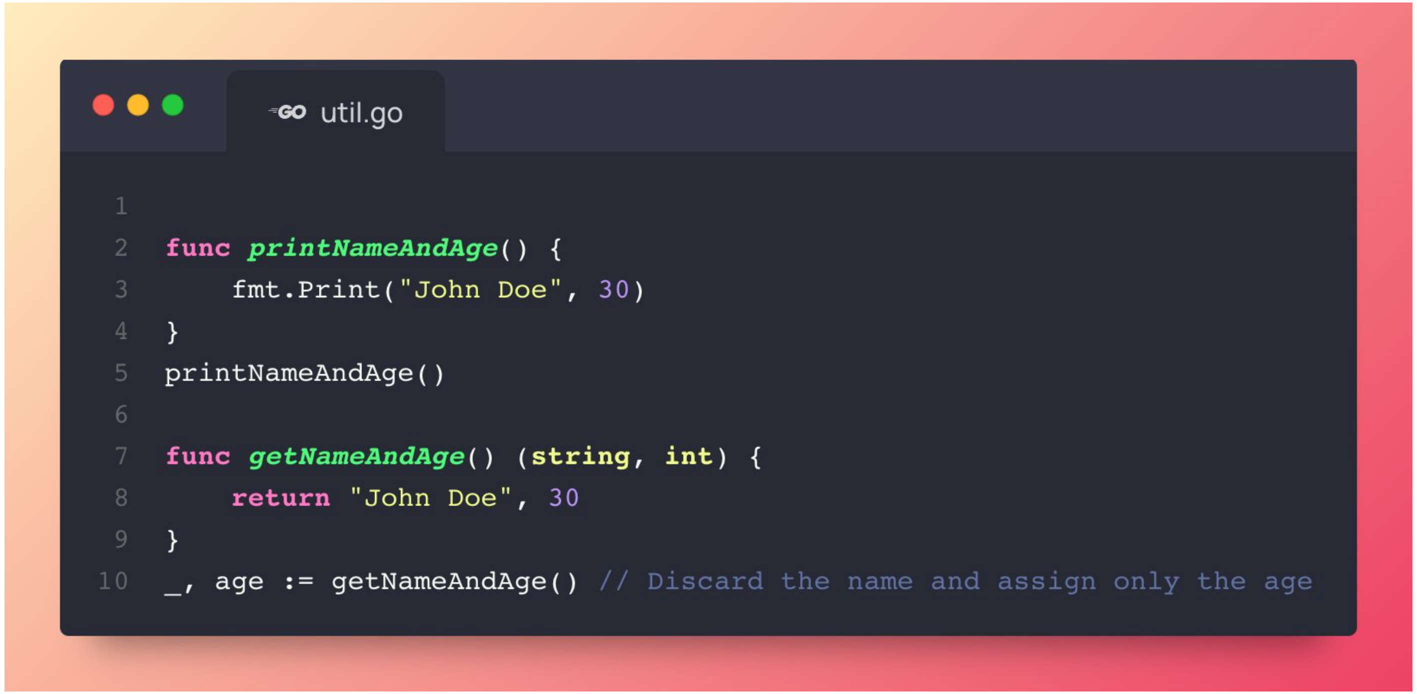
To call a function, use the function's name followed by parentheses ().

value VS reference



```go
func increment(n int) {
    n = n + 1 // Modifying the copy
}

count := 5
increment(count)
fmt.Println(count) // Output: 5

func incrementByRef(n *int) {
    *n = *n + 1 // Modifying the value at the memory address
}

count := 5
incrementByRef(&count)
fmt.Println(count) // Output: 6
```

Return values


util.go

```go

func printNameAndAge() {
	fmt.Print("John Doe", 30)
}
printNameAndAge()

func getNameAndAge() (string, int) {
	return "John Doe", 30
}
_, age := getNameAndAge() // Discard the name and assign only the age
```

Package Basics

Explain the code organization using packages

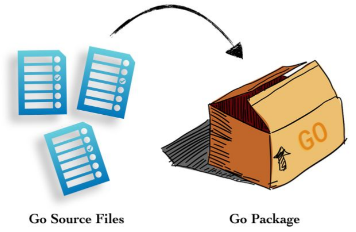
DWARVES FOUNDATION

Package

Packages provide a way to organize code into **reusable** and **modular units**, making it **easier to manage** and **maintain large codebases**.

A package is a **collection of Go source files** in the **same directory** that are grouped together **based on a common purpose or functionality**.

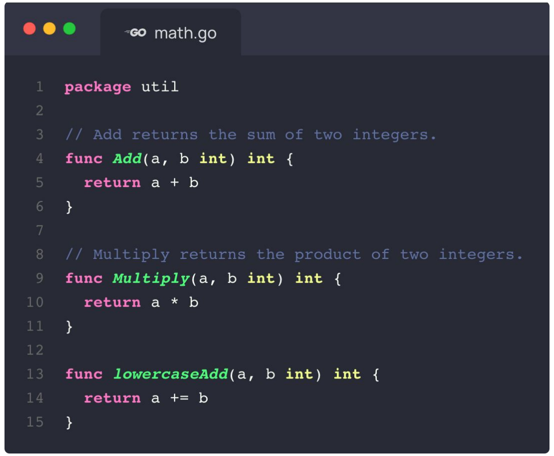
Ex: standard libraries



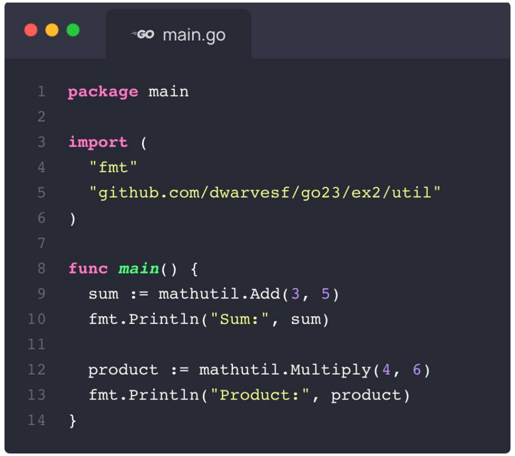
Accessing Package Functions

Exported functions start with an UPPERCASE letter (PascalCase), while unexported functions start with a lowercase letter (camelCase).

Only exported functions can be accessed from other packages.


math.go

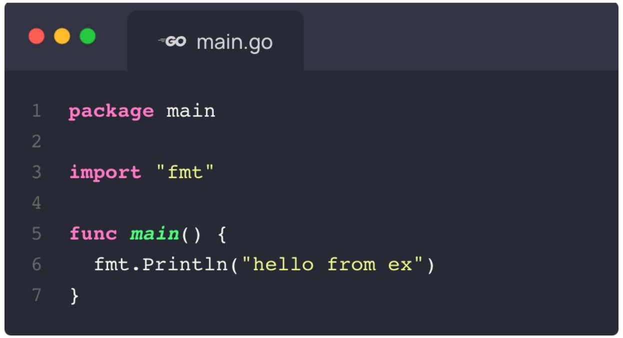
```go
package util

// Add returns the sum of two integers.
func Add(a, b int) int {
  return a + b
}

// Multiply returns the product of two integers.
func Multiply(a, b int) int {
  return a * b
}

func lowercaseAdd(a, b int) int {
  return a += b
}
```


main.go

```go
package main

import (
  "fmt"
  "github.com/dwarvesf/go23/ex2/util"
)

func main() {
  sum := mathutil.Add(3, 5)
  fmt.Println("Sum:", sum)

  product := mathutil.Multiply(4, 6)
  fmt.Println("Product:", product)
}
```

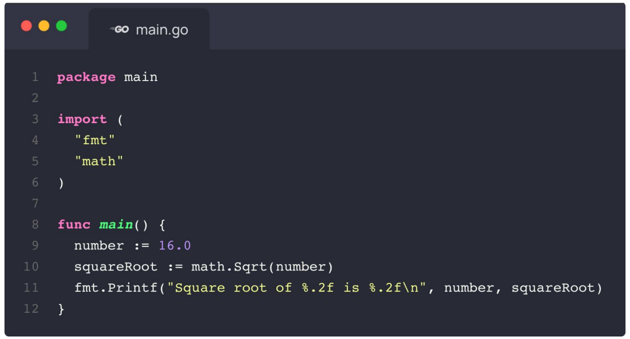
DWARVES FOUNDATION

Importing



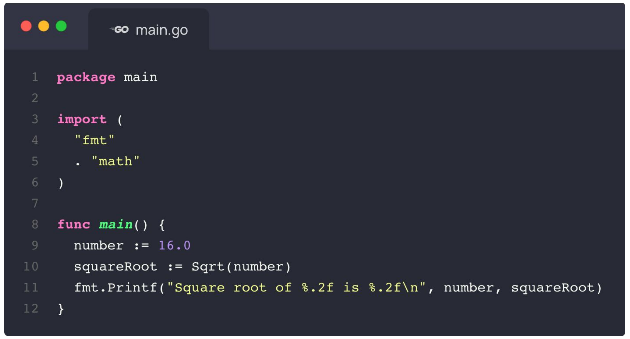
main.go

```go
package main

import "fmt"

func main() {
  fmt.Println("hello from ex")
}
```



main.go

```go
package main

import (
  "fmt"
  "math"
)

func main() {
  number := 16.0
  squareRoot := math.Sqrt(number)
  fmt.Printf("Square root of %.2f is %.2f\n", number, squareRoot)
}
```

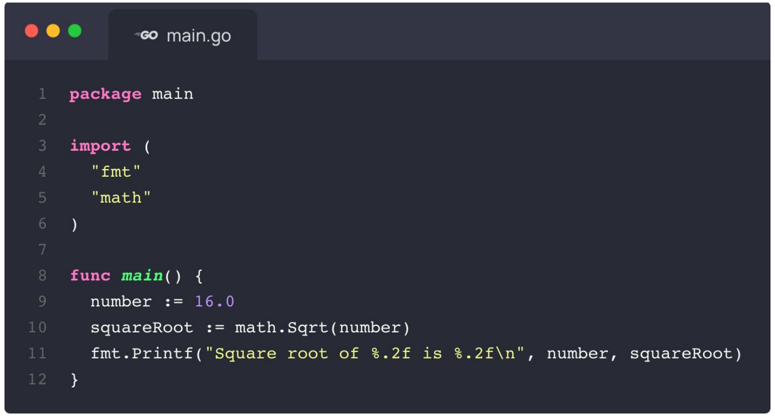


main.go

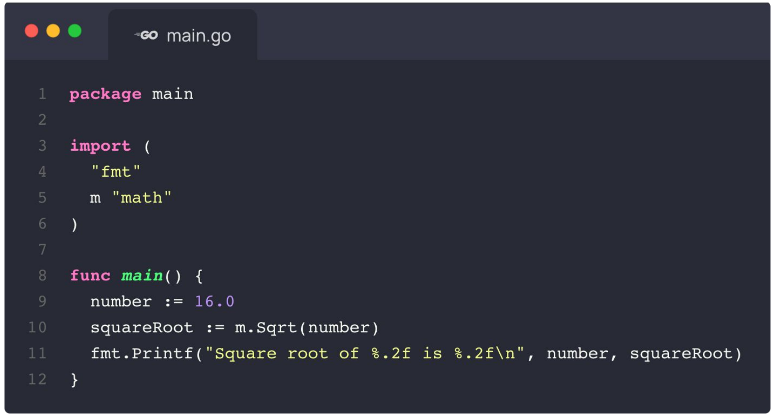
```go
package main

import (
  "fmt"
  . "math"
)

func main() {
  number := 16.0
  squareRoot := Sqrt(number)
  fmt.Printf("Square root of %.2f is %.2f\n", number, squareRoot)
}
```

DWARVES FOUNDATION

Importing with alias


main.go

```go
package main

import (
  "fmt"
  "math"
)

func main() {
  number := 16.0
  squareRoot := math.Sqrt(number)
  fmt.Printf("Square root of %.2f is %.2f\n", number, squareRoot)
}
```

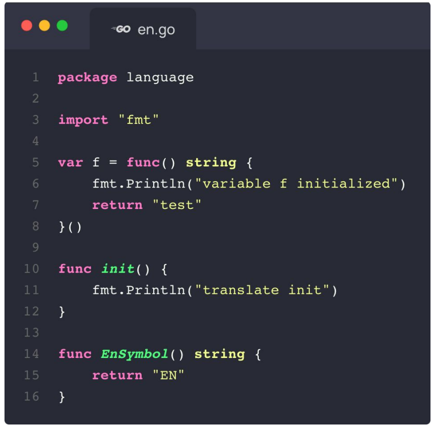

main.go

```go
package main

import (
  "fmt"
  m "math"
)

func main() {
  number := 16.0
  squareRoot := m.Sqrt(number)
  fmt.Printf("Square root of %.2f is %.2f\n", number, squareRoot)
}
```

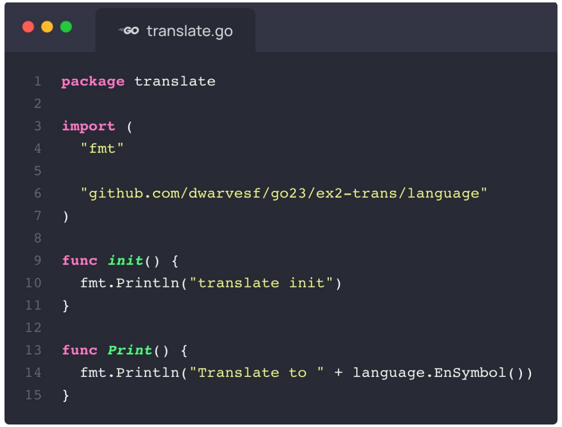
Init function

Go allows the use of an init function in a **package** to perform **initialization tasks**.

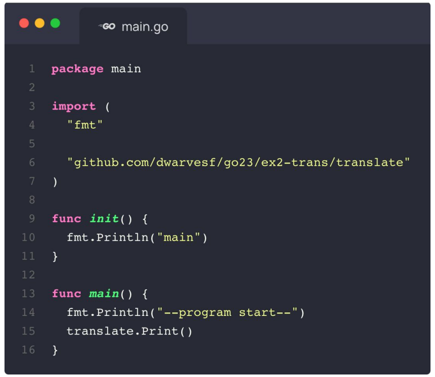
The init function is automatically executed when the package is imported, even before the main function is called.


en.go

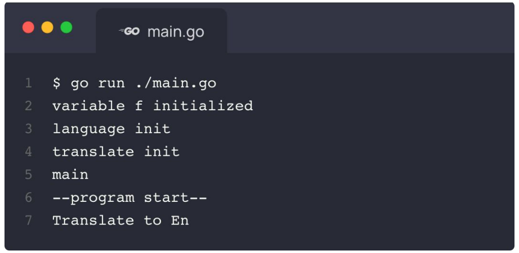
```go
package language

import "fmt"

var f = func() string {
    fmt.Println("variable f initialized")
    return "test"
}()

func init() {
    fmt.Println("translate init")
}

func EnSymbol() string {
    return "EN"
}
```


translate.go

```go
package translate

import (
  "fmt"

  "github.com/dwarvesf/go23/ex2-trans/language"
)

func init() {
  fmt.Println("translate init")
}

func Print() {
  fmt.Println("Translate to " + language.EnSymbol())
}
```


main.go

```go
package main

import (
  "fmt"

  "github.com/dwarvesf/go23/ex2-trans/translate"
)

func init() {
  fmt.Println("main")
}

func main() {
  fmt.Println("--program start--")
  translate.Print()
}
```


main.go

```
$ go run ./main.go
variable f initialized
language init
translate init
main
--program start--
Translate to En
```

DWARVES FOUNDATION

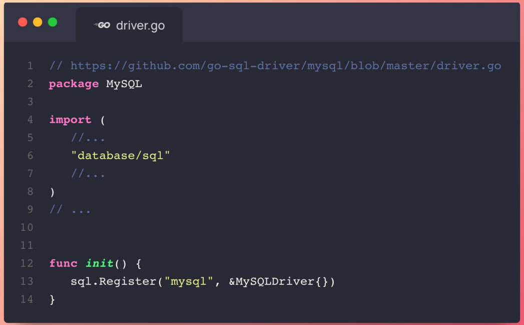
Init function rule

The imported packages are initialized

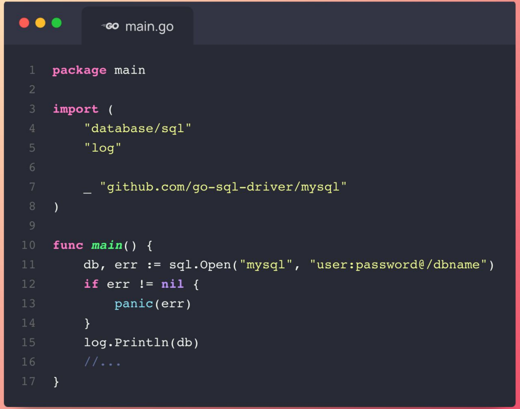
- Variables are initialized
- Init functions are run

Then the package itself is initialized

- Variables are initialized
- Init functions are run



```go
// https://github.com/go-sql-driver/mysql/blob/master/driver.go
package MySQL

import (
   //...
   "database/sql"
   //...
)
// ...


func init() {
   sql.Register("mysql", &MySQLDriver{})
}
```



```go
package main

import (
    "database/sql"
    "log"

    _ "github.com/go-sql-driver/mysql"
)

func main() {
    db, err := sql.Open("mysql", "user:password@/dbname")
    if err != nil {
        panic(err)
    }
    log.Println(db)
    //...
}
```

Practices

DWARVES
FOUNDATION

Practices - function

Keep functions small and focused.

Use meaningful names for functions and parameters.

Follow the single responsibility principle.

DWARVES FOUNDATION

Practices - package

Organize code into packages for modularity and reusability.

Follow naming conventions for packages and files.

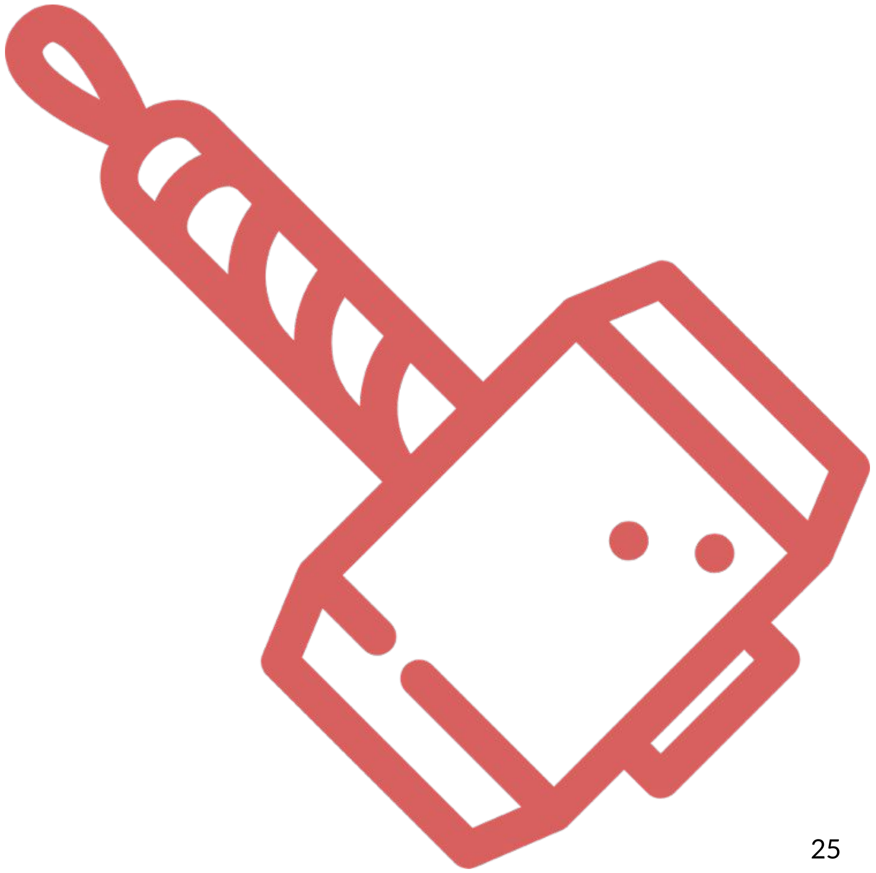
- Use lowercase package names without underscores or hyphens (e.g., "mypackage" instead of "my-package").

Maintain a logical folder structure.

Demo

Demo - Zer0 to Hero

- Set up the basic Go project structure.
- Implement the server's initialization and basic routing.
- Create a simple "Hello, World!" endpoint to ensure the server is running correctly.



DWARVES FOUNDATION

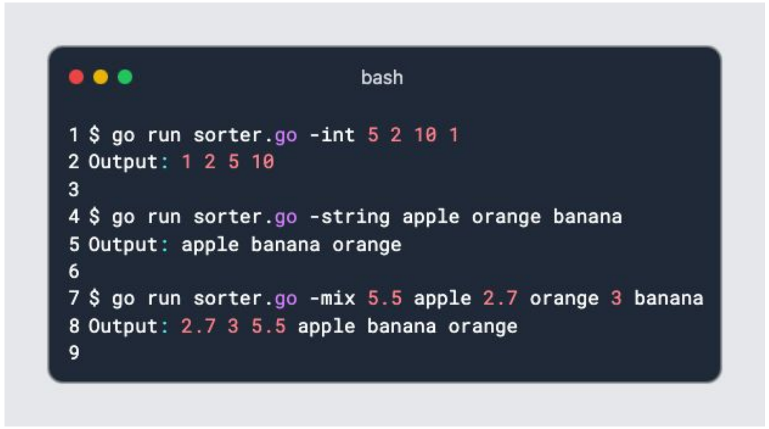
Assignment

Assignment for Day 2

Goal: Create a package and a command-line tool to sort input provided by the user.

Inputs: Number (integer or float) array, string array.

Outputs: Sorted result based on the provided input type.



```
$ go run sorter.go -int 5 2 10 1
Output: 1 2 5 10

$ go run sorter.go -string apple orange banana
Output: apple banana orange

$ go run sorter.go -mix 5.5 apple 2.7 orange 3 banana
Output: 2.7 3 5.5 apple banana orange
```

DWARVES FOUNDATION

Assignment for Day 2

Create a Go package with functions for sorting integer arrays, float arrays, and string arrays.

Implement sorting logic for each data type using appropriate algorithms.

Assignment for Day 2

Create a command-line tool (CLI) to parse the input from the command line.

Determine the type of input (integer array, float array, string array, or mixed).

Utilize the corresponding sorting function from the package to sort the elements.

Output the sorted result.

Assignment for Day 2 - Hint

https://github.com/spf13/cobra

https://github.com/devfacet/gocmd

Use the flag package to parse command line arguments.

Create separate functions in the package for sorting each data type.

Consider implementing generic sorting functions using interfaces to handle mixed input types.

Recaption

Functions: Reusable code blocks that perform tasks in Go.

Parameters: Input values passed to functions.

Return Values: Output values returned by functions.

Packages: Used to organize and share code.

Importing Packages: import keyword to access functions and variables.

Package Aliases: Simplify package references.

Best Practices: Focused functions, meaningful names, and organized code.

Reference

Resources & Reference links

- https://go.dev/tour/basics/1

DWARVES
FOUNDATION

Thank You

DWARVES FOUNDATION

Q&A